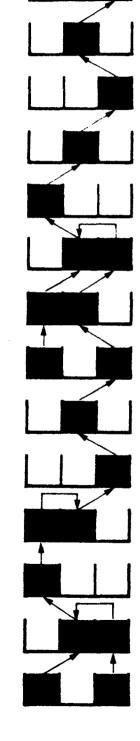
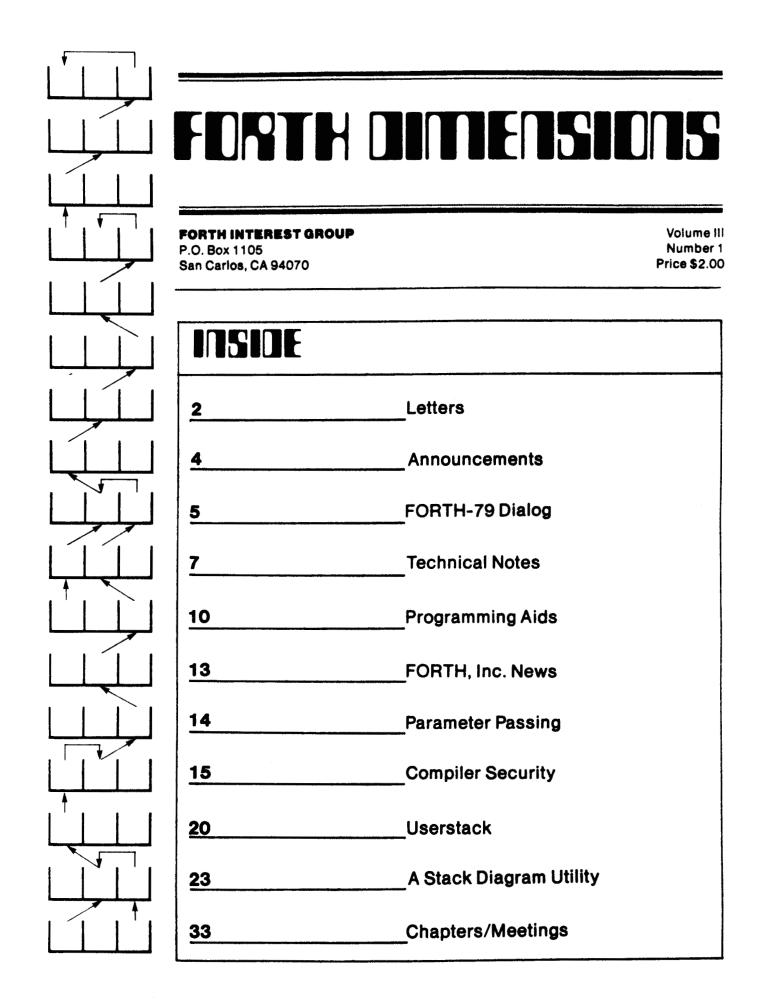
FORTH DIMENSIONS

Forth Interest Group P.O. Box 8231 San Jose, CA 95155

VOLUME III Numbers 1 - 6







Published by Forth Interest Group

Volume III No. 1	Ma
Publisher	Ro
Guest Editor	

May/June 1981 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 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 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 with available programming tools, especially for observatory automation.

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

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

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 WRITER—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. (I'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, FORML, FIG and other applicable "F" 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, INC. 27 East Kings 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 11/5.

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 P.O. Box 218 Fontana, WI 53125

Just goes to show you that there is more than one type of fanatic! Keep 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 much 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. Huttley UNIVERSITY OF AUCKLAND 19 Duncan Avenue Auckland \$, New Zealand

You are very welcome! -- ed.

Editor's note:

At the WEST COAST COMPUTER FAIRE in California two versions of a FORTH bumper strip were circulated:

?FORTH IF HONK THEN

or alternately

: LOVE-FORTH IF HONK 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, OK 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 recommendations.
- 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 O. 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 27th 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 concisely 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 <\def(\overline{\pi}, \overline{\pi})\) operators. What is the precise definition of where the minus sign character is to be stored? Why was this word changed from its former function between <\overline{\pi} and \overline{\pi}\)?</p>
- 3. The word ':' is defined as a nonprecedence 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 definition of VOCABULARY is not clear regarding the possibility of "subvocabularies" such as ABC chained to XYZ chained to FORTH. If this is allowed, and, ABC is the CONTEXT vocabulary, is not ABC, XYZ, and FORTH searched?
- 6. What is the mnemonic 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 transfer 16-bit words ("cells"), while the word CMOVE 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 standard'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 conversations 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 Does the second sentence clear. "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 DOES > ? FIG model or has its definition changed? Taken literally, FORTH-79 says that CREATE will generate an unsmudged header with the CFA pointing to the run time procedure for variables. Is this what is intended?

COMMENTARY FROM THE FORTH DOCTOR

- 1. Some computers apparently (by Standard Team comment) round quotients and remainders to smaller magnitude (more negative). Trucation of negative quotients would do this. If a correct representation is not possible, the result should be nearer zero. Dave Boulton is more knowledgeable on this point.
- " Sign is to be used within < # and # >. The user chooses where to store the sign. Notice that no word generates 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 immediate 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 compilation 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 painstakingly 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 execute only from CONTEXT (and FORTH). No chaining is recognized, beyond context leading to FORTH. 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 Standard. 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 leftovers. 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.

: CPU CONSTANT DOES > ;

is equivalent to

: 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. Kraft

EXPECY OOI (EXPECT MODIFIED FOR UMER DEFINED BACHAPACE CHAR) OO2 (24FE01 TOBY L KRAFT OO3 EXPECT OO4 OVER + OVER (ADD COUNT TO ABOREDS FOR L*** OO5 DD OO6 KEY DUP (GET CHAR AND S*** OO7 OE +ORIGIN # - (GET SYSTE** OO8 IF OO7 DROP DUP I = 010 DUP R> 2 -011 IF ** (ADD COUNT TO ADDRESS FOR LOOP LINIT) DU KEY DUP (GET GHAR AND SAVE COPY) GE +GRIGIN & - (GET SYSTEM BACKSPACE AND CHECK FOR IT) IF CHECK BU CUUST SUFFER POINTEL CE ENDIF (BELL & BEBIN IF 015 LEAVE DROP BL 0 (PREPARE TO LEAVE) 016 ELEE DUP ENDIF 017 I C' 0 I I* (STORE CHARACTER *** 018 ENDIF 019 ENIT 020 LOOP 021 DROP 021 DROP IF DROP DUP I = (LOBE CHAR , CHECK BUFFER BEBIN) DUP R> 2 ~ +>R (ADJUST SUFFER POINTER APPROPRIATELY IF 07 ELBE BSTOF C0 ENDIF (BELL 0 BEDINNING , BS OTHERWIBE) ELSE DUP OD * (CHECK FOR CARRIAGE RETURN) (STORE CHARACTER IN SUFFER) (ECHD CHAR TO TERMINAL) OLDBEARY 001 (GLDBEARY OEMERATOR ROUTINES) 02 DECIMAL 003 0 VARIABLE CHD 004 TOF CR CR 32 SPACES " GLDBEARY" CR CR (GEMERATE PAGE MEADING) 005 " LEN HORD" 15 BPACES " NFA PFA" CR CR , 001 WE' 016 PTERMINAL IF GUIT ENDIF 017 ELSE GUIT THEN AGAIN

Modifications to the fig-FORTH boot-up literals:

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. $C' \setminus '$) to emit for backspace for use on printing terminals.

4

1

i

1

1

1

1

1

1

4

1

1

1

1

- 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'BBFF' two characters of data FF - form feed character (X'OC' initial value) BB - back space character
 - (X'08' 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'07'). 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'.--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:

is read "address under count leaves nothing," and

is read "Numberl under Number2 leaves Number3." 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 KIM, although I don't have my 16K 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 User 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 \$FF. 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 (Vol. 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-itin-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 st th screens doing a text match for the target string. All matches will be listed with the line number and screen number.

Happy SEARCHing!

39 41 SEARCH COUNT	
00 VARIABLE COUNT ER	240
1 COUNT ER +! COUNTER @	4 40
1 COUNTER +! COUNT ER @	4 40
56) IF 0 COUNT ER !	540
12 EMIT 01 TEXT O COUNT ER!	840 OK

SCR # 40

	(SEARCH, OVER RANGE OF SCREENS WFR)
1	DECIMAL
2	00 VARIABLE COUNTER
3	: BUMP (THE LINE NUMBER AND HANDLE PAGING)
4	1 COUNTER +1 COUNTER @
5	56 > IF O COUNTER !
6	CR CR 15 MESSAGE 12 ENIT THEN ;
7	: SEARCH (FRON, TO TARGET STRING)
8	12 EMIT OT TEXT O COUNTER !
9	1+ SWAP DD FORTH I SCR 1
10	EDITOR TOP
11	BEGIN ILINE IF Q N SCR ? BUNP THEN
12	1023 RB 8 < UNTIL
13	LOOP ; CR ." SEARCH IS LOADED " :5
14	
15	TYPICAL USE TO LOCATE 'KEY-WORD': 21 44 SEARCH KEY-W

PROGRAMMING AIDS & UTILITIES

Kim Harris FORTHRIGHT ENTERPRISES P.O. Box 50911 Palo Alto, CA 94303

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

8 : .S (prints stack contents, top last: stack unchanged) DEPTH IF SP@ 2- SO @ 2- DO I ? 9 -2 +LOOP 10 ELSE "Empty" THEN : 11 12 (which vocabulary is being referenced?) 13 : .VOC (prints CONTEXT VOCABULARY name) 14 CONTEXT @ 4 - NFA ID. ; 15 SCR # 82 0 (Tools: number printing fig-FORTH 1.x) 1 2: .BASE (-) (prints current radix in decimal) 3 BASE @ DUP DECIMAL . BASE ! : 4 5 (create base-specific stack-print operators) 6 : BASED. (EUILDS: newBase -) (DOES): n -) BUILDS 7 8 DOES > @ BASE @ SWAP BASE ! SWAP . BASE 1 ; 9 10 16 BASED. H. (print top-of-stack in hex) 11 8 BASED. O. (print in octal) 12 2 BASED. B. (print in binary) 13 14 15 CK

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

HEX : +INDEX 113 0 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

0 ('VIEW' USING 'WHERE' 4/15/81 R.E.E.)
1 (ADAPTED FROM FORTH DIMENSIONS VII,
NER 6, P 162)
2 FORTH DEFINITIONS
3: DOC (BLK @, IN @ ,; (SAVE BLK AND
DISPLACEMENT)
4 : CONSTANT \geq LOC < [COMPTLE] CONSTANT ;
(REBUILD THE WORDS)
5 : VARIABLE > DOC < (COMPTLE) VARIABLE ;
(THAT BUILD WORDS.)
6:: >DOC < LOOMPTLE1: ;
7 - BUILDS >DOC < [COMPILE] BUILDS ;
8 : USER > DOC < [COMPILE] USER ;
9 : CREATE DOC (COMPTLE) CREATE ;
10 : VIEW [COMPILE] ' NFA (GET HEAD OF
THE HEADER)
11 DUP (COPY THE ADDRESS)
12 2 - Q (GET THE DISPLACEMENT)
13 SHAP 4 - @ (GET THE BLOCK NUMBER)
14 WHERE (GO SHOW & HIGHLIGHT);
15 ;S

HELP WANTED

Senior Level FORTH Programmers

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" 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 Z-80 computer (S-100 bus, 6 MHz) The results of a few bench marks were:

Program : LOOPTEST 7FFF 0 DO LOOP ;	<u>Timin</u> 2.3 sec	<u>Z-80</u> 2.9 sec
: -TEST 7FFF 0 DO I DUP - DROP LOOP;	5.9	7.4
: *TEST 7FFF 0 DO I DUP * DROP LOOP ;	44.0	54.9
: /TEST 7FFF 0 D0 7FFF I / DROP LOOP	; 74.3	88. 6
: WIPE 120 61 DO I CLEAR LOOP ;	34.3	81.8
97 LOAD (four hundred eighty 9's)	17.9	18.6
\$		

I was surprised that Timin-FORTH which is 8080 fig-FORTH ran faster than Z-80 FORTH which uses the extra Z-80 registers for IP and W. Dr. Timin's opinion was that the 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 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 Z-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.

3

3

3

٦

3

:

£.

2

;

3

5

h O

s

s

e e

Έ

Ή

ď

:h .n

١t

in

:/1

My overall impression 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 Z-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 (tm, 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".

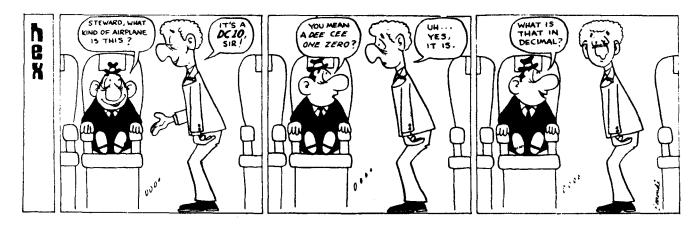
Our hope is that half of the callers will review the available material and then ask questions. The other half should add answers to these questions. You simply find a topic or message and attach your query/response. Users naturally organize their material in a form that facilitates retrieval.

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

For availability contact:

The CommuniTree Group Box 14431 San Francisco, CA 94119

or call the original Tree: (415) 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 distributors in Japan, software 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

P/

1

just been completed for Work has 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 programmed independently, are using This is a capability they've polyFORTH. 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 Zimmerman, 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 RK05 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 Sygnetron 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 programming in FORTH.

: DELAY 0 DO LOOP ; (SPIN FOR A WHILE)

100 DELAY (COUNT PASSED ON THE STACK)

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 + Rate (0 + (0); (PUSH THE N'TH PARAMETER)

: EXAMPLE < BUILDS DOES > R# ! 1 \$ 2 \$ EXECUTE ; EXAMPLE ZZZ 90 , 'EMIT CFA , (TYPE A "Z") EXAMPLE SPC 10 , 'SPACES CFA , (TYPE 10 SPACES)

Now that the mechanics are explained the following example will more fully demon-

strate its usage. Both DUMP (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 DUMP and CDUMP.

: U.R O SWAP D.R;

: DUMPS (BUILDS DOES) Hof ! (STORE PARAMETER BASE ADDRESS) BASE @ R HEX (SAVE BASE AND SET HEX) OVER + SWAP (CONVERT TO BEGINNING AND END ADDRESS) BEGIN CR LUP 4 U.R 2 SPACES (TYPE ADDRESS) 1 \$ 0 DO DUP 2 \$ EXECUTE 3 \$ U.R 4 \$ + 2DUP = OVER 16 MOD O= OR DF LEAVE THEN LOOP 20UP = ?TERMINAL OR UNTIL DROP DROP CR (RESTORE BASE) R) BASE ! ; DUMPS COUMP 16, 'C@ CFA, 4, 1, (2-ADDRESS, 1-COUNT)

DUMPS DUMP 8, '@CFA, 6, 2, (2-ADDRESS, 1-COUNT)

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. 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 programming errors made by the programmer 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 programming 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 THEN, DO LOOP, etc.) leave as IF 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 IF, 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 BEGIN	1 1	UNTIL WHILE	(1) 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		+/L00P	(3)			

Page 15

Ī

នរ :1

Ci

1

21

:

i: t:

O.

v

P

3

P

7

£1

1

s

3

:

:

:

i

a e

đ

2

ť

P ť

Ċ

P 5

ti c 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-EXAMPLE flag — true or false message) IF ." True part 1 " ELSE ." 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" messages will print. To borrow a phrase from Kim Harris, probably "<u>Not</u> 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 ?PAIRS expects by executing ?PAIRS . to find a matched pair of values, in this case ones (1), as a matched set of conditional 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 The word "probably" is used completed. 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:

[SCREEN 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 state to execute properly. These words are programmed to contain one of the following words:

?COMP ?EXEC ?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 determine 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 BLK indicates that input is coming from the user's terminal and that the machine is therefore not loading.

The 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 (#22) occurs. whi

221

eti

sti

de:

77

CO

(🔹

T2

th

۰.

1

(🛛

si

21

(.

c(f]

f]

tı

ť

t:

1

ð

ť

fi

ť

b

C

£

c

ĩ

۹

i

1

1

1

1

1

١

1

i

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 !CSP, ?CSP, ?PAIRS. These words are used to protect against structural programming errors as described above in STRUCTURAL ERRORS. An explanation of each of the uses of these words is as follows:

ICSP ?CSP

These words are used together to check for changes in the stack position. !CSP stores the current stack position in ?CSP compares the user variable CSP. to the current stack the value in CSP position and, if they are not the same, issues a Definition Not Finished error !CSP and ?CSP are currently (#20). 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 defini-For example, if a definition tions. should have no stack effect (leaves the same number of items on the stack as it removes) the following would test this:

!CSP cccc ?CSP

which would execute a definition named accc 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 correct structure in compiler directives (see STRUCTURE ERRORS) to check that the value of the two numbers on the stack is the same. If the value of the two compiation conditionals on the stack is not the same, a Conditionals Not Paired error #19) occurs. ?PAIRS can be used to test similar situations in user programs, but the error message given will be the same (error #19).

The checks on the dictionary and stack consist of testing the stack for underflow, the dictionary and stack for overflow, 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

τ

ı

5

¢

5

Ľ

У

5

ê

y

n

g

d

-

n

e

t

1

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 (#4). 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 (#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 unpredictable 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 S0 to check for a stack under-It also checks that there are at flow. least 128 bytes of dictionary space left (to leave room for PAD and stack work). If the stack underflows an Empty Stack error (#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 programmer, though it is executed frequently during compiling and text interpretation.

WORD (A-FORTH 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 Or just when we think we need it? time? 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 pos-The system should not suffer the sible. overhead for those who do not desire it. If security is desired, a "Novice Programmer Protection" package could be compiled into a user'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 Definitely, also the us when necessary. 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 often incomplete and annoying. Anyone who has tried to write and secure a good structure, or a BEGIN general CASE REPEAT loop which allows mul-WHILE tiple 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. Additionally, 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 1105, San Carlos CA 94070 and tell us what you think.

NEW PRODUCTS

POLYMORPHIC FORTH NOW AVAILABLE

FORTH is now available for the Poly-Morphic 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:

1

1

1

1

1

1

i

1

1

1

1

1

ī

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 util-User receives both 16k and 32k ities. versions with user space being 2k and 18k 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 Those familiar with FORTH users only! should have no trouble using this system there is no manual inluded). (i.e. sufficient references are However, 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 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 The motivation for this violation data. 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 calculate 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 Note: 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 MY-ARRAY. 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 (@) 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 STACK MY-STACK

MY-STACK 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-STACK

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 PUSH 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 As discussed pushed or popped differs. earlier, STACK deals with integer (two byte) words. DSTACK 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-STACK and ?STACK. 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-STACK so that it will be empty. Again using the MY-STACK example,

?STACK MY-STACK

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, STACK, DSTACK, and FSTACK are all defined in terms of a more general NSTACK defining word. A line such as:

22 4 NSTACK WIDE-STACK

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. 1. 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

2382

ADDRE

COMMENTS

	1	CHAR. COUNT	
ĺ	n	CHARACIERS	# characters saved
			for word name
i	2	VOCAB LINK	
	2	CODE ADDRESS	Points to ;CODE part of NSTACK def.
	2	CURRENT STACK PIR	
	1	#WORDS/STACK ELEMENT	
1	2	MAX SIK PIR ADDRESS	
	2	MIN SIK PIR ADDRESS	
	m	STACK DATA AREA	# bytes needed to
			contain specified #
			of stack elements

Figure 1 -- Dictionary Layout for a Stack Type Variable *** BLOCK 0 150

- (STACK DATA TYPES -- PHH 23 OCT 80 3 COLLEGATES CONCEPTION AND A COLLEGENESS OF A COLLEGENESS STACK
- (BTACK)
 > NETACK (BUILDE SHAP OVER DUP + + DUP (BEYTES IN STACK)
 NERE + DUP , (BET UP CURRENT BTK PTR)
 NOT C. (MONDE/STK BLEN) , (NAX BP ADDR)
 NOT C. (MONDE/STK BLEN) , (NAX BP ADDR)
 NOT C. (HIN SP ADDR) 10 + ALLOT (BAPACE POR STACK)
 (CODE MPAANA 2 + B LXI, B DAD, H PUBH, (NL PTS TO MARDE/ELE)
 NP UALD, H DCX, D H HOV, H DCX, E H HOV, AP ANLD, (SAV IP)
 H POP, B POP, (PUBH/POP FLAG) C A HOV, B OBA, O=,
 IF, (POP DATA FROM NSTACK TO PARAMETER STACK)
 H AOV, (MARDE/ELE) H DCX, H B HOV, H DCX, H C HOV,
 (COP IN BC)
 -->

*** BLOCK # 151

(NSTACK DEFINITION CON'T) BEGIN. (MOVING DATA, MORDE AT A TIME) PSM PUBN, B LDAR, B INK, A E MOV, B LDAR, B INK, A D MOV, PSM POP. (COLNT) D PUBN. (DATA FROM NETACK) A DCR. O., A DCR. 0-, END, SE. (PUBH PARAMETER STACK DATA TO NETACK) H A NOV. H DCX. H B HOV. H DCX. H C HOV. (COP IN BC) SEGIN. (TO PUSH DATA) D DOP. PSM PUSH. (COUNT) B DCX. D A HOV. B STAX. B DCX. E A HOV. B STAX. PSM POP. A DCR. 0-, END. ELSE. THEN. C M MOV. M INX. B M MOV. (SAVE NEW COP) R LALD. M E MOV. M INX. M D MOV. M INX. RP SHLD. NEXT JAP.

*** BLOCK # 152

(STACK TYPE VARIABLES CON'T) CDDE POP 0 H LTI, FUBH JMP, CDDE FUHH -1 H LTI, FUBH JMP, I STACK 1 HISTACK 1 (INTEGER ELEMENTS) I DSTACK 2 HISTACK 1 (DOULLE PHEC INTEGER ELEMENTS) I FSTACK 3 HISTACK 1 (FLOATING POINT ELEMENTS) · USER AIDS WHICH CANNOT BE COMPLIED ...) EMPTY-STACK NOON - EMPTIED THE USER STACK "BOOM" BY (RESETING ITS STACK POINTER) EMPTY-STACK >/(DUP 3 + 0 SMAP 1) (?STACK NOON -- PRINTS OUT THE CONTENTS OF THE USER STACK) ? STACK NOON -- PRINTS OUT THE CONTENTS OF THE USER STACK) ?STACK NOON -- PRINTS OUT THE CONTENTS OF THE USER STACK) PRINTS 2000 7' USER STACK EMPTY - THEN 1 1.8

٨υ ÛK

OK DK 100 STACK MY-STACK OK 35 STACK YOUR-STACK OK 11 22 33 44 55 66 77 88 99 OK PUSH MY-STACK OK PUSH MY-STACK OK PUSH YOUR-STACK OK PUSH MY-STACK OK PUSH MY-STACK OK PUSH MY-STACK OK PUSH YOUR-STACK OK + POP MY-STACK --11 OK. OK POP YOUR-STACK POP YOUR-STACK 2DUP . . + . 77 33 110 0 OK POP MY-STACK . 110 OK 55 OK OK POP MY-STACK . 66 OK OK POP MY-STACK . 88 OK OK POP MY-STACK . 99 OK OK ЭK ж 0K OK. 0K OK ?STACK MY-STACK USER STACK EMPTY OK OK 11 PUSH MY-STACK 22 PUSH MY-STACK ?STACK MY-STACK 3A7A 0016 000B OK ж UK EMPTY-STACK MY-STACK 7STACK MY-STACK USER STACK EMPTY OK OK OK ж OK.

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 -1flag 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 (flag=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 REMARKS

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 It is felt that the concept, in bounds. 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 computers that combines Fig-FORTH with standalone 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 available on one 5-1/4" disk for Cl (Superboard), C2 and C4 machines with 24K. Product includes a structured 6502 macroassembler 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

DEA- EDI---

I AM AFR--- THA- THE LET--- IN THE LAS- ISS-- ABO--FOR-- INC- USI-- ONL- THR-- LET--- NAM- FIE--- HAS HAD THE UPP---- EFF--- FRO- WHA- THE WRI--- WAN---

FIS LET--- LIA- THI- ONE) SHO-- THA- SAV--- NL-THP-- LET--- AND COC-- IS JUS- ABO-- OPI---- L: TER-- OF A TRA-- OFF BET---- SAV--- MEM--- AND KEE---- LEG------

WE STI-- DON-- SEE THE NEE- FOR 31 CHA----- NAM--

YOU-- TRU--

Cha Moo

CHU-- MOO--FOR-- INC-

P.S- MR. FRE- 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 straightforward. 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-Thus, DUP, OVER, DROP as well as lent. many other primitives would be coded before I started. Even as it was being bu ca

ge

01

V2

TE

ad

if

of

cc

10

C'S

3

9

.

E

be

24

51

1

3]

te

\$1

25

20

fr

fi

vi

s y

fr

8

TR

Cđ

li

it

20

th

25

fr

36

fi

da

20

-

Page 23

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 IF 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# DOC defname	to document 1 definition
screen# SDOC	to document a whole screen
screen# PRIDOC	to print from given screen
PDOC	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 It prompts "DROP?" to see are unknown. 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 program 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 encountered, 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 "OK" 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 FORTH 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?				
ANALY ZE	addr	len		
SWAP	len	addr		
INCH	1			
DROP?				
PUSH VALUE?	char			
PUSH VALUE?				
INCH	llen	addr	char	
DUP	llen	addr	char	char
78	len	addı	char	char 7F
-	len		char	(char-7F)
IF	len			
DUP	llen		char	char
0 D	llen	addr	char	char OD
-	len	addr	char	(char-0D)
IF	llen	addı	char	
DUP	llen	addr	char	chai
OUCH				
DROP? 1				
PUSH VALUE?				

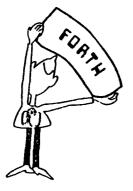
FIGURE 1 (cont.)

OUCH OVER Cl	len len len	addr addr addr	-	addr
1+		(add)		
SWAP		r+1)		
1-	• • -	r+1)	•)
ELSE	llen			
(llen		char	
DROP	len	addr		
SWAP	addr	len		
DROP	addı			
	addr	20		
OUCH	1			
DROP? 1				
PUSH VALUE?				
OUCH	addr	-		
0	addr	-		
OVER	addr	0	addı	
CI	addr			
0	addı	-		
THEN	addr	-		
ELSE (l len	addr addr		
-			Char	
DROP 8	len len	addr addr	8	
•	i ien	acor	0	
OUCH DROP? 1	1			
PUSH VALUE?				
OUCH	i len	addı		
1-	•	(add	r-11	
SWAP		(aud r-1)		
1+			(len+)	1
THEN			(len+]	
;		r-1)		
о к	, (230	/		

ANALYZE	laddt len
	len addr
	len addr char
	len addr char char
71	len addı chai char 77
-	len addr char (char-7P)
IF	l len addı char
	llen addı char char
	l len addı char char OD
]]en addr char (char-0D)
	llen addr char
	ilen addı char char
	ilen addı char
	len addr char addr
	len addı
	llen (addr+1)
	(addt+1) len
	(addr+1) (len-1)
	len addı char
	i len addr chai i len addr
	iaddı len iaddı
	addt 20
	i addi 20
	addr 0
	i eddi 0 addi
	addi
	0 Ibba
	addr 0
	l len addr char
	len addr char
	len addr
8	llen addr 8
OUCH	llen addr
	len (addr-1)
	(addz-1) len
	(addz-1) (len+1)
	(addz-1) (len+1)
•	(addr-1) (len+1)
OK	

FIGURE 2

OK 100 LIST SCR # 100 0 U : ANALYZE SWAP INCH DUP 7F - IF DUP 0D - IF DUP OUCH OVER CI 1+ SWAP 1-ELSE (CR) DROP SWAP DROP 20 OUCH 0 OVER CI 0 THEM ELSE (DELETE) DROP 8 OUCH 1- SWAP 1+ THEM ; 45 6 08



HELP WANTED

FORTH PROGRAMMER

Entry Level - Will Train

John Sackis Data Breeze 2625 Butterfield Rd. Suite 112E Oakbrook, IL 60521 (312) 323-1564

FIGURE 3

PDOC STACK DIAGRAM - SCREEN # 100

0

8

v t

W

Э

3 e

P

T

1

1

S

ų,

c P

W

3

1......

¢ ¢

1

-

1

¢ ŧ

1

đ

1

ī

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. It may later be displayed or printed by PDOC.

The spool file contains the encoded screen from which the diagram was made followed bv variable length lines separated by carriage return characters. The file is terminated by an ascii null character. It resides on a set of consecutive screens. The first screen and maximum number are determined by literals in SPIT and PRTDOC. I use 10 screens starting at 230. These may be copied elsewhere and printed by PRTDOC. Failure to copy them will cause the listing to be lost the next time a function or screen is Hagrammed.

IMPLEMENTATION PROBLEMS AND SOLUTIONS

It is important not to search the standard vocabularies when diagramming stacks. This is because actions are different for the same name, depending upon state. By way of example, for the operator + must concatenate the symbolic name strings representing these elements with an embedded plus sign, rather than adding the top two elements on the Also, not all operators are stack. 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 in fig-FORTH, this option is searched. not directly available since the FORTH vocabulary is searched after the current This may be solved by vocabulary. 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.

wasn't obvious until T+ the implementation began that operators would require concatenation of their identifying It was also decided that strings. parenthesis would be placed around each level of expression nesting so that ambiguity could be eliminated without rearranging expressions for precedence. This occasionally leads to expression such 88 ((array+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 diagramming 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 BLOCK will have to be rewritten if this is not the case in your system. I recommend that you instead generate a new system with 1k 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 go 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 PSTAK. 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 useful 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

'FIND pfa length tru	e (found)
----------------------	-----------

-- 'FIND false (not found)

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 diagrammed input string.

stl st2 -TEXT cond

True if the two strings differ.

val 1- val-1

Decrements the top of stack value by one.

vl v2 2DROP ----

Drops the top 2 elements off the stack and discards them.

--- :: ---

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 diagrammed.

--- :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.

--- :SP ----

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.

1

4

i

.

t

ť

C

i.

d

f

---- 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 diagramming 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 src+1 src

Intermediate function to set up for MVDEL.

src 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#.

--- PSTAK ----

ş

5

1

è

I

3

2

:

ł

٢

r

9

đ

С

33

d

y

1

Prints all words from the string addresses on the stack. The top element is printed to the right of previous elements. The stack is unchanged.

adr cnt PWRD -

Prints one word via MTYP. Used by PSTAK.

--- REPEAT ---

Functionally identical to the redefined THEN.

scr# 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.

scr# SDOC ---

Documents one whole screen by executing it, using the diagram definitions.

--- SKBD ---

This scans the keyboard for user interaction. It generates the "DROP?" and "PUSH VALUE?" prompts. It is invoked whenever intervention is required in the ilagramming process.

char SPIT ---

Writes character out to disk spool file.

---- STACK ----

This is the name of the vocabulary containing 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.

```
102 THIAD
 SCH # 102
0 { stack diagram package 1 of 14 b. A. Cole 810326)
          : 1- 1 - ;
: 2DROP DROP DROP ; -->
               Calling sequences:
screen DOC defname
      2
                     screen SDOC
    7
10 DUC builds stack diagram for one definition.
11 SDOC builds stack diagrams for entire screen.
12 PDOC prints stack diagram built by DOC or SDOC.
     13
14
15

      SCR # 103
      0 ( stack diagram package 2 of 14 B. A. Cole 810326)

      10 VABIABLE SPOOL ( offset into spooled print file )
      0 (all file into spooled print file )

      20 VABIABLE SPOOL ( pointer to current input word )
      0 VABIABLE STU ( pointer to current input word )

      30 VABIABLE SUM ( in -TEXT: true for difference )
      0 (all file number being printed )

      50 VABIABLE SUM ( in -TEXT: true for difference )
      0 (all file number being printed )

      60 VABIABLE SC ( current screen # spooled diagram)
      0 vABIABLE IFTA ( address of top of IF stack )

      90 VABIABLE IFST 20 ALLOT ( define IF stack )
      10 .FST IFFTR ; ( and initialize to empty )

      11 --2
      13

     13
14
15
  SCR # 104

0 ( stack diagram package 3 of 14 B. A. Cole $10326)

1 SII SPOL # 1024 /MOD 9 MIN 230 +

2 BLOCK + CI 1 SPOL +1 UPDATK ; ( spool one character )
           : ISP SPOOL # 0 SPIT SPOOL 1 :
                                                                                                              ( mark spool file end )
        5 1 DEPTH SO # SP# - 2 / 1 - ;
                                                                                                               ( compute stack depth )
       8 : STYP SWAP DUP BOT + SWAP DO
9 I # SPIT LOOP 32 SPIT 32 SPIT ;
                                                                                                               ( apool word
                                                                                                                                                                         )
           : NTTP 2DUP STIP TIPE 2 SPACES :SP ; ( type and spool word )
     11 : HIT 2007 SITE ITE 2 SPACES :SP

12

13 : PWRD DUP C@ 72 >

14 IF .: ERR * ABORT

15 ELSE COUNT MTYP THEN ; -->
                                                                                                               ( print encoded word )
   eeses from: Barry A. Cole Los Angeles, CA 213-390-3851 *****
```

```
SCR # 105

0 ( stack diagram package = 4 of 14 = B. A. Cole = B10326)

1 : PSTAK DEPTH IF SP# 2 - SO # 2 - ( print all words on stack )

2 BEGIN DUP # PWHD 2 - ZDUP = EMD

3 DROP DROP THEN ;
    CI SWAP 1+ ALLOT ;
                                                                                     ( concatenate 2 words )
    13 -->
    15
SCR # 106

O ( stack diagram package 5 of 14 b. A. Cole 810326)

1 : 7NUM DUP 10 C# 95 s. ( test if input is number)

2 BECIN 10 DUP C# 16 DICIT

3 WALLE DAOP REFEAT C# 32 *:

4 : HAED CR 13 SPLT 10 SPLT DUP ( print line header)

5 COUHT 10 MAN SWAP OVER MTTP 11

6 SWAP - 0 DO 32 SPLT SACE LOOP

7 .* 1 = 124 SPLT 32 SPLT :SF ;

6 : MVB ROT DUP 10 SWAP; ( setup for MVDKL)

9 : MVDEL BECIN MYB C# 3R NVB I SWAP CI ( sec dst 1en -- )

10 ROT DUP R) = HDR ROT ZDROP; ( seve t to a til delim)

11 : G-HERE HERE PAD HERE 10 ON MYDEL

2 HERE - 10 DUP 10 - SWAP DUP

13 IF HERE CI ALLOT

14 : ELS ZDROP ZDROP C THEM ;

15 : ESC PAD C# 27 = IF .* ESC - ABORT THEM ; ( escape ) -->
   SCR # 107
           ( stack diagram package 6 of 14 B. A. Cole 510326)
VOCABULARY STACK O CONSTANT LLIMIT ( filled in later)
       î
            - SKBD CR .- DROP7 - KEY DUP EMIT ( scan kbd for drop,pushes)
               AB IOR DUP 9 (
IF DEPTH 1 - MIN 0 DO DROP LOOP
ELSE DROP THEN
BEGIN CR .. PUSH VALUE? • PAD 80
EXPECT ESC G-HERE 0+ END ;

    :: NFD ?: # 7NUN
    ( handle word not found)

    :F ?: #
    ( number)

    ELSE SKDD SPOOL # SPL 1
    ( undefined)

    :: # :HEAD DROP SPL # SPOOL 1 THEN ;

      10 : :NFD 11 # 7NUM
      12
13
14
               14
    ***** from: Barry A. Cole Los Angeles, CA 213-390-3851 *****
   O.K
   108 THIAD
   SCR # 108
       0 ( stack diagram package 7 of 14 b. A. Cole 810326)
1 : FIND -FIND DUP IF 2DNUP DUP LLIMIT >
2 IF 1 ELSE 0= THEN THEM ;
       3
       BECIN FIND HENE DUP DUP TI 1
                                                                                     { stack diagram redef of t)
{ word to mem and printed}
             5
    IN A REPEAT

SCRST O SPOOL I DUP 256 /HOD

SVAP SPIT SPIT ;

3

14 : SDOC SCRST STACK INC.
               7STACK PSTAK REPEAT ( print stack)
DROP PSTAK CR 13 SPIT 10 SPIT :SP SP! ;
                                                                                    ( reset spool ptr and place)
( screen# in spool)
                                                                                        ( document 1 screen)

      SCR # 109
      0 ( stack diagram package B of 14 B. A. Cole 810326)

      1 : PHDG 12 EMIT
      ( print heading on top of page )

      2 .* STACK DIAGRAM - SCREEN # "

      3 :SC 7 CR CR 54 :LN 1;

            : PRTDOC DUP :BK I BLOCK # ( print diagram from spec scr)

:SC I PHDG 10240 2 DO

I 1024 /MOD :SK # * BLOCK * C# -DUP

IF DUP EMIT 10 *

IF PLOP EMIT 10 *

IF PLOP THEN THEM

ELSE LEAVE THEN LOOP ;
        3
       0
       12
13 - PD00 230 PRTDOC ;
                                                                        ( print last generated diagram)
```

SCR # 110 O (stack diagram peckage 9 of 14 B. A. Cole \$10326) 1 : -TEXT O SUM I SWAP O (true if 2 strings differ) 2 DO VER I + C# OVER I + C# - SUM +1 LOOP 3 2DROP SUM # ; 4 5 : WTEST OVER DUP CP 1+ SWAP -TEXT ; (test 2 words for ()) 6 7 : DOC SCRST HERE 32 WORD DUP C# 1+ ALLOT (document 1 def) 8 BLK # >R IM # >R 0 IM + SWAP BLK + 9 BEGIM 58 WORD IM # >R 32 WURD (find colon,word) 10 HERE WTEST 11 WHILE #> 1024 + 2 IF -* NOT FOUND* ABORT THEM REPEAT 13 DROP R> IM + STACK :: (now go from here) 14 [COMPILE] FORTM R> IM + R> bLK + ; 15 --> now go from here) ***** from: Barry A. Cole Los Angeles, CA 213-390-3851 ***** 111 TRIAD SCR # 111 O (stack diagram package 10 of 14 B. A. Cole 810425) 1 : PSH 2 IPPTR +1 IPPTR @ 1 ; 2 : :SAV HERE :PSH SP@ DEPTH 1 - DUP , 3 DUP + HERE SWAP DUP ALLOT CHOVE ; 4 : :RST IPPTR @ @ DUP @ DUP 2 @ ROT + SWAP -DUP IF 5 0 D0 DUP @ SWAP 2 - LOOP THEN DROP ; 6 : :FILL -2 IPPTR +1 : 5 6 : :KIL IKILL -2 IFPTR +1 ; STACK DEFINITIONS HERE 'LLIMIT 1 7 9 --> ۱ó 11 12 13

15

SCR # 112 0 (stack diagram package 11 of 14 B. A. Cole 810425) 9 : ELSE DEPTH 128 + >R SPI :RST R> IFPTH ₽ € 1+ CI ; ۱ó 11 : IF DHOP :SAV ; FONTH DEFINITIONS --> 12 13 14 15 SCR # 113 O (stack diagram package 12 of 14 B. # 1 : GBLD IN I 32 WORD HERE C# 1+ ALLOT; 2 : GWRD IN # (BUILDS GBLD 3 DOES>; 4 GWRD) GWRD (5 : G(1) IN # (BUILDS GBLD 6 DOES> SWAP) (SWAP :C B. A. Cole 810326) (SWAP :C :C : 1.4.12 7 : G(2) IN @ <BUILDS GBLD B DOES>) :C :C 9 : G(3) IN @ <BUILDS GBLD 10 DOES> SWAF) :C :C (SWAP :C ; 10 DOES> SWAP) :C :C :C (SWAP :C ; 11 : (41 WORD ; IMMEDIATE --> 12 13 14 15

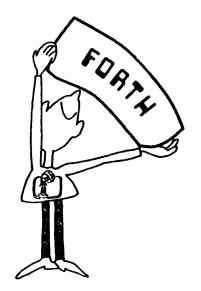
esees from: Barry A. Cole Los Angeles, CA 213-390-3851 ***** OK

34 15 ars

105 THIAD

FORTH CLASS

INA THIAD
5CH # 114 0 (stack diagram package 13 of 14 H. A. Cole #10425) 1
<pre>2 STACK DEFINITIONS 3: 32 WORD HERE DUP C# 1+ ALLOT : 4: DUP DUP; : ROT ROT; : SWAP SWAP; : DVER OVER; 5: R> DROP; : DROP DROP; : . DROP ROP; : RSG; 6: DECIMAL DECIMAL; : 2 DUP ZDUP; : 1 DROP DROP; : C1 ;</pre>
7 : +1 !; I, DROP; I SPACE; I (; I); 8 : 21 I DROP; I DROP; I ALLOT DROP; I BLANK !; 9 : C, DROP; I DO !; I DUMP !; I ENIT DROP; I -DUP DUP; 10 : END DROP; I ERASE !; I +LOOP DROP; I /LOOP DROP;
1): EXPECT 1; : MOVE 1 DROP; FLEAVE; 12: SPACES DROP; : TYPE 1; FBEGIN; FLOOP; : \; FCN; 13: ENDIF THEN; REPEAT THEN; FWHILE IF; 14: AGAIN; FUNTIL 1; 152
SCB # 115 C (stack diagram package 14 of 14 B. A. Cole 810425)
3 GWAD 0 GWRD 1 GWAD 2 GWRD 3 2 G(1) - : MINUS - ; G(3) - G(3) + G(3) 0 G(3) / 3 G(3) 4 : AND 4 ; G(3) 1 : OR 1 ; G(3) 5 : XOR 5 ; 4 G(2) +1 : 1+ +1 ; G(2) +2 : 2 + +2 ;
5 G(2) +2 : 2* +2 ; G(2) /2 : 2/ /2 ; G(2) -2 : 22 ; 5 G(1) NOT GMED I GWED cond " GWED here : HERE here ; GWED pad : PAD pad ; 5 : = 2DROP cond ; : ALLOT DROP ;
<pre>9 : VARIABLE DROP GWRD ; : CONSTANT VARIABLE ; 10 : < = ;</pre>
13 14 15
5CR # 116 0 2
8 9 10
11 52 53 13 14
eress from: Barry A. Cole Los Angeles, CA 213-390-3851 eeses It



Date:	June	22 -	· 26
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: \$100 \$140 plus room and board
- How: Call Prof. Ron Zammit (707) 826-3275

MMS-FORTH FOR STRINGY FLOPPIES

Kalth 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:

> Kalman Fejes KALTH MICROSYSTEMS P.O. Box 5457, Station F Ottawa, Ontario K2C 3J1 Canada

How to form a FIG Chapter:

- 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 x 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

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

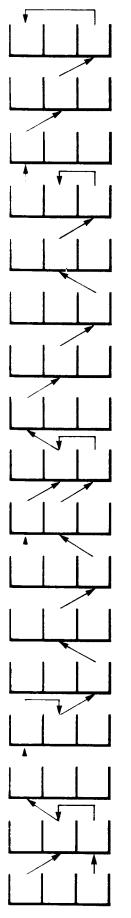
- 3rd Wed MMSFORTH Users Group, 7:00 p.m., Cochituate, MA. Call Dick Miller at (617) 653-6136 for site.
- Seattle Chuck Pliske or Dwight Vandenburg at (206) 542-7611.
- Potomac Paul van der Eijk at (703) 354-7443 or Joel Shprentz at (703) 437-9218.

Tulsa Art Gorski at (918) 743-0113.

- Texas Jeff Lewis at (713) 719-3320 or John Earls at (214) 661-2928 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, GU16 5AU, England
- Japan Mr. Okada, Presdient, ASR Corp. Int'1, 3-15-8, Nishi-Shimbashi Manato-ku, Tokyo, Japan.
- Canada Quebec Gilles Paillard at (418) 871-1960.
- 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.



t

ז ב

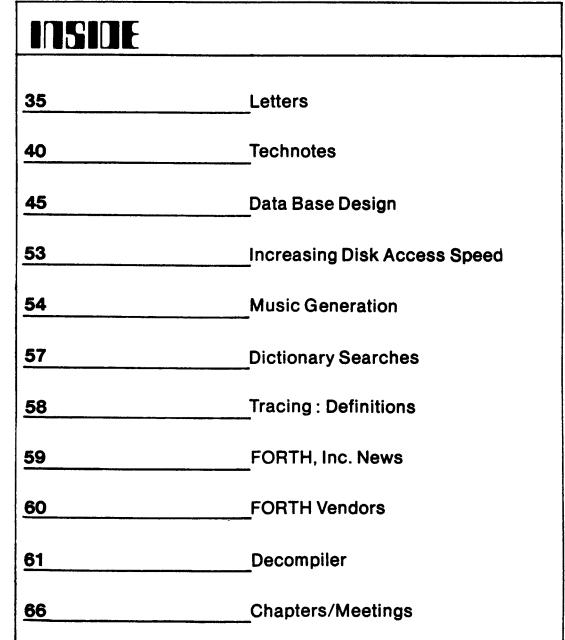
t

ĩ

FORTH DIMENSIONS

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

Volume III Number 2 Price \$2.50



FORTH DITTERSIONS

Published by Forth Interest Group

Volume	III No.	2	

July/August 1981

Publisher 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 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 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

HISTORICAL PERSPECTIVE

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

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

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

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 have 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. If 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 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

fi

I

a

t

LETTERS

Dear FIG,

Please find enclosed two short articles which might be suitable for publication in FORTH DIMENSIONS.

I did not ask for the publication kit, so I hope the articles do not violate your rules too much. Second, my native language is not English but Dutch, so forgive me if there are any errors and feel free to correct them.

Please note my new telephone number and correct it in your listing of local FIG chapters.

We have not had many meetings lately, probably because our members are too active!

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

Thank you for your articles, Paul. Readers can find them under the applications area of this issue.--ed.

Dear FIG,

I recently purchased a listing of fig-FORTH for the 8080 from you and I am very impressed with the Language package. You will find enclosed an order and Bank Draft for several books which I eagerly await. I received my Dual Micropolis Mod II Disk trives only two weeks ago and my first project was to assemble FORTH. The disk interface routines were quite easy to link to the Micropolis DOS using ideas from the CPM interface supplied. However, when I tried to LOAD a short word definition off a screen the system would lock up and not :ome back with any error messages or the OK'; because the system would compile words from the keyboard and the Disk I/O worked well, I was puzzled as to why there was 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 ENCL1 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. Keep 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, SCR#200 line 8 should apparently be 08 CASE for left curson as opposed to OB as printed, since OB 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 introducing 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.

21

7

æ

ť

۲

鸖

r

đ

ť

6

a

3

ī

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 8K 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 benchmark 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 user. Suffice to say that FORTH is by definition a disk based system. If you do not have a disk then you are compromising on an area vital to obtaining the real potential of the system.

Regarding benchmarking, it has always been the position of FORTH DIMENSIONS that the nature of FORTH makes benchmarks more of an indication of the speed of a CPU than any particular system and we generally do not publish them. This has been discussed at length in previous editions.--ed.

Dear FIG,

While I cannot disagree with the intent of "An Open Response" in FORTH DIMENSIONS. Vol. II, No. 6, concerning the hardware requirements for FORTH, I feel you may discourage 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 8K of RAM and traditional cassette-storage. My "terminal" was a memory-mapped 16-line by 32-character display with ASCII keyboard. This minimal system has given me hours of pleasure and practical experience with FORTH, and because of the concise nature of FORTH has been capable of powerful constructs. An acquaintance has installed a cut-down version on a 5K 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-FORTH 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, BRK 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 320x200 dot raster-scan display, and I am interested in corresponding with others concerning FORTH-based graphics processors.

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

The point of the "Open Response" was not to condemn 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 Ol 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 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 DIMENSIONS. 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' dish radiotelescope which I also use for looking at thunderstorms.

I am slowly getting together parts of a Western Digital-based computer. Their pcode 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.

Z,

t

-

0

fi

1.1

Don Latham Six Mile Road Huson, MT 59846

OK 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.

Dear FIG,

l

2

3

2

2

3

n

3

5

A little note about changes in the situation in NW Europe. During the second half of March, there was an exhibition in Malmo (close to Copenhagen) - "Datacraft Sl" (Computer power-81).

Up until then, FORTH was very difficult pet in touch with here in Sweden. To y great astonishment, there were at least perhaps 6, systems running in different pols. The most interesting one was a poly-FORTH system running on an ABC-80 (a V Z-80 lowend machine). There were also fig-FORTH's running realtime setups on PET's.

To me, who had up until then been 'dryswimming ' FORTH, it was quite an experience 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 HP 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 unparalleled the user is allowed Please do not insult us by freedom. 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 column, 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.

FORTH, Inc. has the following job openings:

TECHNOTES, BUGS, FIXES

TIPS ON BRINGING UP 8080 Fig-FORTH

Ted Shapin 5110 E. Elsinore Aenue 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-\$.

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 FFFØ AND 17 ?ERROR SCR @ (LINE) DROP ;

: -MOVE LINE C/L CMOVE 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., O P (Mini-editor)

1

will place a comment on line 0 of the current screen.

Type the rest of the lines above and then use the word "FLUSH" to write the mini-editor to disk. Now, when you need to start the system again, just type 85 LOAD and your mini-editor will be put into the dictionary.

Use the mini-editor to type in the Fig-FORTH editor. The string search screen can be omitted if you do not have a version written in highlevel FORTH.

USING ENCLOSE ON 8080

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

	UB .	8711	ENCLOSE
	DB	'ENCLOS	
	DB	'E'+80H	
	DW	PFIND-9	
E 101	DW	\$+2	
	POP	D	;(DE)<-(SI)=DELIMITER CHR
	POP	H	;(IIL)<-(S2)=ADDR. OF TEXT TO SCAN
	PUSH	EI .	:(S4)<-ADDR.
	MOV	A.E	;(a) <- DELIM CHR
	LXI	D,-1	; INITIALIZE CHR OFFSET COUNTER
	DCX	Ħ	;(HL)<-ADDR-1
			SKIP OVER LEADING DELIMITER CHRS
- SCL1	INX	51	
	INX	D	
	CMP	11	; IF TEXT CHR - DELIN CHR
	J7.	ENCLI	THEN LOOP AGAIN
			ELSE NON-DELIM CHR FOUND
	PUSH	D	;(S3)<-(DE)=OFFSET TO IST NON-DELIH
	PUSH	PSW	SAVE DELIM CHR ON STACK
	HOV	А,Н,	; IF 1ST NON-DELIM-NULL
	ANA	A	
	JNZ	ENCL2	
	POP	PSW	THEEN DISCARD DELIM CUR
	INX	D	;(S2)<-OFFSET TO BYTE FOLLOWING NULL
	PUSH	D D	;(SI) <- OFFSET TO NULL
	PUSP	D	(SI) -OFFSEI ID NOLL
	JMP	NEXT	
CCL2	POP		VANCORT IN CUR FROM STACK
		-	
	PUSII		SAVE DELIM CHR ON STACK
	MOV		
	ANA	Α.	
	JNZ	ENCL2	THEN CONTINUE SCAN
			ELSE CHR-NULL
	POP	PSW	DISCARD DELIM CHR
	PUSH	D	;(S2)<-OFFSET TO NULL
	PUSI	D	;(S1) -OPPSET TO NULL
	JMP	NEXT	
			ELSE CHR-DELIM CHR
ENCL4	PUSH	D	;(S2)<-OFFSET TO BYTE FOLLOWING TEXT
	INX	D	;(S1) -OFFSET TO 2BYTES AFTER END OF WORD
	PUSII	D	
	JMP	NEXT	
	INX INX CMP JZ PUSII MOV ANA JNZ POP PUSII JMP PUSII INX PUSII	ENCL2 PSW D D NEXT D D D D	ELSE CHR-NULL DISCARD DELIM CHR (52) (-OFFSET TO NULL (S1) (-OFFSET TO NULL ELSE CHR-DELIM CHR (52) (-OFFSET TO BYTE FOLLOWING TEXT

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 Fig-FORTH cross-compiler, METAFORTH, by John Cassady should be noted:

page 26 screen 66 line 7 should read

KISR H LXI SRA5 SHLD 12 ORG + LHLD 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 METAFORTH 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 fig-FORTHs from their respective vendors.

L

Ì

E

3

5

t

)

12

CHANGING 8080 fig-FORTH FOR DISK COPYING

Ted Shapin 5110 E. Elsinore Avenue 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 SPDRV1 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 Since the current definition is done? 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 The Standard Team has sugthis area. gested one mechanism for evolutionary "Experimental changes in FORTH via 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 1st 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 16K 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: 16K of ram

Manual:

ł

r

2

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 SYM 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 SATURN SOFTWARE LIMITED 8246 116A Street Delta, B.C., V4C 5Y9, CANADA (604) 596-9764

OSI-FORTH 2.0 / FIG-FORTH 1.1

This is a full implementation of the FORTH 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 MF, C8P DF, C3, C2-8P 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 they are produced. Letters added **as** discussion Twenty-four pages of of particulars for OSI, utility screens, and operation of the editor (includes sample FIG Installation manual edit screen). 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

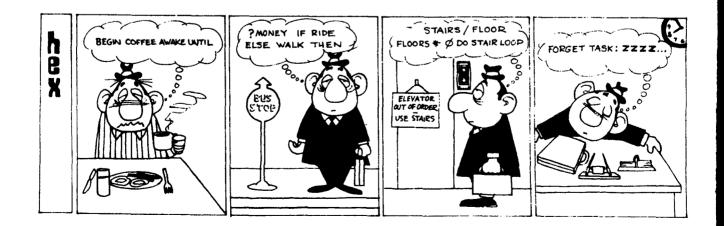
DATRICON FORTH

Datricon now offers D-FORTH, a software package designed for use in conjunction with microprocessor-based, STD Bus compatible 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 communications 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.

re

D

8-

rd 5f

le

js

or

ut

ar

es

ŧ

Ĺ

2

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.

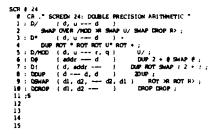
The first two Screens provide eight utility FORTH words for developing a data base file. The comments included in the Screens within parentheses should, combined with the mnemonic nature of the words, give you a clue to what is happening. The first two words are variables used in manipulating the file, 'REC#' and 'OPEN'. '20' is a FORTH word, and alias for 'DQ', which fetches the next two values beginning with the address on the top of the stack and places them on the The word, 'LAYOUT', places two stack. 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 con-The word 'RECORD' takes the sideration. value for a record number from the stack and returns its address to the stack.

Finally, 'ADDRESS' takes the record number at the variable 'REC#' 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.

	¥ 23	
	CR ." SCREEN 23: FI	LE DEVELOPHENT - 2 "
1	: TFIELD (BUILDS)	Create a test field)
2	OVER DUP . +	leaves file count for this definition ;
3		leaves: addr. count)
4	28 ADDRESS . SHAP	:
د	DFIELD BUILDS	create a data field)
6	JVER +	.eaves file count for this definition "
-	DOES>	leaves address
8	ADDRESS + .	
9	. FILE	Create a named storage allocation i
12	< BUILOS .	origin block)
	1.	number of records in file)
12	CUP B/ BUE JVER	<pre>/ * , / # bytes per block)</pre>
- 2	, t	# bytes per record)
14	DOES' DEEN	when file name used, point to i
	:5	its descriptor parameters.)
154	¥ 15	

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 currently 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 In subsequent use, that newly a name. 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 'REC#' to be left on the stack, but also the length of that field. 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

TI Ľ

Ē

3

5

2

s

ε

3

a

Ъ

t

e

11

f

Ъ

Ъ

•

i

c

a

₽

Ъ

Ъ

i

▲ 5

t

24

e

đ

U

a

h

2

e

f

ť

than 128 bytes, but with a larger block size, larger records can be used. Tn 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 handling dollar amounts and storing them as double precision integers.

SCR	1 25
e	CR . SCREEN 25: DATE CONFRESSION AND EXPANSION "
1	: DATEBIT (converts date input to 2 bytes)
2	32 D* D+ 13 D* D+ DROP ;
3	: ?DATE ." (HHV/DD/YY) "
- 4	QUERY 47 WORD HERE NUMBER 47 WORD HERE NUMBER BL WORD
5	HERE NUMBER DATEBIT ;
6	: .DATE (formats 2 byte date code and prints)
7	8 13 U/ 32 /MOD ROT 18C * ROT + 8 100 D* ROT & D+
8	() # # 47 HOLD # # 47 HOLD # # #> TYPE :
9	;S
10	
11	
12	

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 In the meantime, the present dates easy. 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 The word 'DATEBIT' stack. 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

@ CR .* SCREDH 26: ?SAMDUNT AND .SAMDUNT *

1 ( define action for each scale case )

2: 05CALE 100 D*; : ISCALE 10 D*; : 25CALE;

3: 35CALE .* NNUML REMOR * CR ;

4 ( define scale case and extend for wach with 'CFA' )

5 '05CALE CFA ...

6 '13CALE CFA ...

7 ( scale craw RATABLE MSCALE * ISCALE CFA ...

6 '13CALE CFA ...

7 ( scale double practision value according to 'DML' )

8: SCALE DFL 0 1 MIN 2 * NSCALE * 0 EXECUTE;

9 ( walt for decimal value and scale 1t - leave value on stack )

10: ?SAMDUNT JUERY 8L WORD HERE MUMBER SCALE;

11 ( print d from stack as 5 and right justify in 0 spaces )

12: .SAMDUNT

13: CLP ROT ROT DABS v4 0 4 46 HOLD RS SIGN 4>

14: 36 ENIT DUP 0 SMAP - SPACES TYPE;

15: S

NSG 0 15

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

13

1/2

n

1

y

ρ

n

ρ

e

e

r

e

σ

W

e

f

e

a

ne

Гo

eτ

у,

re

ng

e.

he

r s

le en

st

he

ew

of

TH

st

28

of

en,

zer

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 'ISCALE' 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, I now find that it is more contoo. venient 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		27			
0	C	CR .	* SCREEN Z	7: DEMO	FILE - RECORD GENERATION "
1	8	2	DFIELD 1	AG (a tag)
2		30	TFIELD N	AME (item name)
3		2	DFIELD D	AY (the date)
- 4		4	DFIELD D	OLLAR (a dollar amount)
- 5			200 (numb	er of r	ecords) 400 (starting block)
6			FILE DEMO)	
7	:	119	ME (wait	for na	me then store it in record)
			NAME DROP	30 32 F	ILL QUERY 1 TEXT PAD COUNT
9			NAME ROT N	IN CHOW	E UPDATE ;
19	:	. N	ME (prin	t name	field) NAME TYPE ;
- 11			(the	rest fo	llow in the same way)
12	:	10	Y TOATE D	YIUPD	ATE : : .DAY DAY @ .DATE ;
13	:	1D	LLAR ?SAMO	UNT DOL	LAR DI UPDATE ;
-14	:	.D	LLAR DOLL	R DE .S	AMOUNT :

15 : .REC CR RECE @ 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

1

1

would be most desirable especially in a hostile environment.

Now, to examine the actual example of the definition of a file which we will call 'DEMO'. Each record will begin with zero offset from the record address and a '0' is entered followed by '2' for a two byte length of a data field to be named TAG'. Many occasions in later manipulation of records make it desirable to have such a field for adding flags, Following this definition, the etc. length 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 field 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 With this, the 4 fields within amount. 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 Then, we use the word 'FILE' to use. 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 In a similar manner, we define '.NAME'. '!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 en-We then define '.DAY' coded date there. to output the date stored in the 'DAY' We get the 16 bit value stored field. 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 In a similar manner'.DOLLAR'is field. 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 FORTH 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 and 'ADDRESS' followed by the 'READ' 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

1

3

3

2

g

d

r

t

-

d

t

h

2

1

0

-

£

빌

fı

3

ti

مة

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

@ CR ." SCREEN 28: DEMO FILE - CLEAR.DATA, INPUT, CUTPUT "

1 ( clear especially tag in the 0 record in file )

2 : CLEAR.DATA 0 READ TAG 128 0 FILL UPDATE ;

3 ( example of formatting for input )

4 : INPUT 0 READ TAG 0 1+ UPDATE DUP TAG ! READ

5 CR CR ." ENTER NAME --> " !NAME

6 CR ." ENTER NAME --> " !DAY ( has a format prompt )

7 CR ." ENTER NAME --> " !DALAR

8 .REC FLUSH ; ( save this record on disk )

9 ( list files 1 through the number in TAG of 0 record )

10 : OUTPUT 0 READ TAG 0 OUP 0= IF CR CR ." EMPTY FILE "

11 DROP ELSE 1+ 1 DO FORTH I READ .REC LOOP ENDIF CR CR ;

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 particular 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 I 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 which means that there are no 'TAG' 'O READ' places the records present. 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 copy is stored back in the field of 'TAG' in the 0 record which is updated. Then

the second copy is placed in the variable 'REC#' 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 utilities used to define a new data base file and the example example of handling data provides some elaboration of the information in-This will be a cluded on the Screens. 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 When the '79 Standards become Model. 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 ultimate use in a wide variety of implementations of FORTH cannot be overemphasized.

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

APPLICATION NOTE:

These FORTH routines have been developed a FORTH OPERATING SYSTEM for the EEATHKIT H89. This system is available from the MOUNTAIN VIEW PRESS, Box 4656, Mountain View, CA 94040. The compiled FORTH program image can be saved on disk and will be up and running in less than four seconds from a cold boot. The system cas 1024 byte blocks which also increases the speed of operation.

However, after develoment, the Screens were loaded on a FORTH implementation ferived from the fig-FORTH FOR 8080 ASSEMBLY SOURCE LISTING which is available from the FORTH INTEREST GROUP, Box 1105, San Carlos CA 94070, in printed form and ilready on disk also from the MOUTAIN VIEW FRESS. This version has 128 byte block and operates in conjunction with CP/M. To this has been added the fig-EDITOR from the fig-FORTH INSTALLATION MANUAL and a single extension, DUMP, used to illustrate the appearance of the records as stored in a block.

The printed session illustrated was made using the CP/M control P to echo the sutput on the printer. The session starts with CP/M loaded and its usual prompt. The CP/M file, FORTH60.COM, is the object module of the fig-FORTH Model. The warning messages are not on Screens 4 and 5 and the warning flag is turned off. Then, the Screens for the fig-EDITOR and a good dump routine are loaded. Finally, the Screens discussed are loaded. The file 'DEMO' is called and the application :f some of the file utilities is illustrated. 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 MANUAL, GLOSSARY, MODEL, Forth Interest Group, Box 1105, San Carlos, CA 94070.

fig-FORTH FOR ASSEMBLY SOURCE 8080 LISTING, Forth Interest Group, Box 1105, San Carlos, CA 94070.

```
a>
a>forth&f
BOBD fig-FORTH 1.1
OK

Ø WADING t ( Werning messages not on Screens 4 5 5 )
UK
OK
OK
47 LOAD (fig=
RMSG # 4 IMSG # 4
FLUSH MSG # 4 OK
                  ( fig-EDITOR )
                  ( My version of a good dump )
49 LOAD
SCREEN 49: GOOD DUNP
0K
0K
21 LOND
                  ( Loads Screens discussed )
```

(fig-fORTN Hodel)

This demonstration data system provides a pattern for the further development of any type of data base. The basic file formating definitions are on Screens 22 and 23. Some utilities are on 24, 25, and 26. The demo file definitions are on 27. Elementary file menjulation utilities are on 28. This model should get you started.

ENTER 'Y' TO LOND SCREENS SCREEN 22: FILE DEVELORIBRY 20 MGG 0 4 SCREEN 21: FILE DEVELORIBRY - 2 SCREEN 26: DOUBLE FRECISION ARITMETIC SCREEN 26: DOUBLE FRECISION ARITMETIC
SCREDN 25: DATE COMPRESSION AND EXPANSION SCREDN 26: 7AANCUNT AND .SAACUMT SCREDN 27: DEMO FILE - RECORD GENERATION SCREDN 28: DEMO FILE - CLEAR.DATA, DNPUT, OUTHUT OK
OK OK
DENO OK Clear.data ok Cutput
DIFTY FILE
OR INFUT
ENTER NAME> 1201TH ENTER NATE> (HAVDD/YY) 4/21/01 ENTER ANOLAT> 18.50 1 ZENITH 64/21/01 \$ 18.500K OK DRPUT
DFTER HAVE> TBH DFTER DATE> (H4/DD/YY) 4/21/81 DFTER ANOUNT> 68. 2 IBH> 64.21/81 \$ 68.040K 0K 0K 104/7

			Ļ
			1
1		_	
		-	
-		L	
1			1
		-	
			•
1			
		5	
			l
			1
		÷.	
			ľ
		-	
1			
1			

et

đi

a

fi

80

åe.

đi

84 81

Ch.

co

be

pe:

6.....

-

a 21

žní

6.

FORTH DIMENSIONS 111/2

3 DEC		
OK OK OK BREAD ADDRESS HEX 88 DUMP DECIMA SSCA 3 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 8 8 8 8 SSCA 3 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8	4 8 8 8 8 9 0 6 8 8 8 8 8 0 0 5A 45 44 8 20 29 20 20 20 20 20 20 20 20 20 20 20 6 6 3A 7 6 49 42 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20	
OK OUTPUT 1 ZENITH 2 IBM 3 DEC OK OK	04/21/01 \$ 18.50 04/21/01 \$ 60.00 04/21/01 \$ 103.50	
OK OK STATEMENT CR OR 20 SPACES ." ST CR OR ." TOTAL VALUE " 33 : DO I READ DOLLAR DE D+ LOOP OK STATEMENT	SPACES 0 0 0 READ THE 0 1+	1
STATEMENT		
1 ZENITH 2 IBM 3 DEC	04/21/01 \$ 18.50 04/21/01 \$ 68.00 04/21/01 \$ 103.50	
TOTAL VALUE	\$ 182.00	
OK		

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 Interest Group seminar, meeting, workshop, or other event is going to be? The fig-Tree 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. Use any 300 or 110 baud modem, and type several carriage returns; then the system is self-instructing.

HELP WANTED

FORTH PROGRAMMERS 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 special systems programming and writing proposals, setting projects: technical specifications, customer liaison, hands-on programing, and supervision of senior programmers. Extensive FORTH programmin; experience, some scientific or engineering 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 CP/M and has then used 8080 fig-FORTH will have noticed that CP/M is much faster than fig-FORTH when reading or writing data on floppy diskettes. The reason for this apparent speed difference lies in the manner in which CP/M stores its files as opposed to how fig-FORTH stores its screens. (Editor's tote: Speed is also reflected in hardware details such as interleaved formatting and direct memory access. It is not necessarily a FORTH characteristic.) I shall attempt to explain the difference.

A single-sided 8" diskette formatted in the normal manner contains 77 tracks, with each track containing 26 sectors with 128 sytes of data in each sector. In order for the disk controller to be able to find # particular sector in a given track, teader data is stored on the diskette just prior to each 128 byte data block - a sort : preamble. Among other information in this preamble is the sector number. A format program writes this information on each track in a consecutive manner; in ther words, immediately following the index hole pulse is sector 1,2,3, ... 26.

A program that reads a sector must first select the proper track and proper sector, then must read that sector's data and store it someplace for use. It is fairly easy to select the proper track and sector and read the data; the problem iomes in trying to read two consecutive sectors. There is not enough time between the time when the first sector's data is read and the time when the next sector is available, to store the data from the first sector and request the data from the This means that reading second sector. consecutive sectors 5 and 6, for example, requires a minimum of two revolutions of the diskette.

CP/M accesses files faster than fig-FORTH accesses screens because the files are not stored in consecutive sectors. CP/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 CP/M is able to read another sector. So instead of storing a file in sectors 1,2,3 ... 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 2 3 4 5 6 7 8 9 10 11 12 13 New format: 1 14 10 23 6 19 2 15 11 24 7 20 3

 Old format:
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26

 New format:
 16
 12
 25
 8
 21
 4
 17
 13
 26
 9
 22
 5
 18

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:

h

o

_

e

е

е

Ħ

h

٠t

۱S

١g

١g

.-

ıd

3.

2,

r –

1.

	Old	New	Speed
	Format	Format	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 simulating interleaveing by this program. The multiple screen copy definition used for the second test is listed in screen 167.

```
SCR # 167
```

```
0 (BATCH-COPY FROM #, TO #, DEST # )
     1
     2 : BATCH-COPY ( FROM-SCR, TO-SCR, DEST-SCR ---- )
     3
                  EDITOR
     4
                  ROT ROT ( DEST, FROM, TO
                                                                                                                                     )
                              SWAP ( DEST, TO+1, FROM
     5
                  1+
                                                                                                                                     )
                  DO
     6
     7
                                               (DEST, I
                            I
                                                                                                                                     )
     8
                            OVER
                                            ( DEST, I, DEST
                                                                                                                                     )
     9
                            COPY
                                            (from acreen I to DEST
                                                                                                                                     )
   10
                            1+
                                               ( DEST+n
                                                                                                                                     )
   11
                    LOOP
  12
                    DROP FORTH ;
  13
  14
  15
ok =
SCH # 51

    4 State
    6 Music - Experimental Constructs
    4 (Music with Digital Newserch: Computers 5-
2 ( Sound Effects hoard - Uses two AY3-891P
    5 sound generator ICs

                                                                                                              852483-888
    5 ( 1/0 port and variable declaration
                                                                                                                                   )
       HEX
            X
AFF CONSTANT REG-PORT
ORF VARIABLE VOICE
AFF VARIABLE BMASKI
POR VARIABLE ALVI
1946 VARIABLE DVAL
                                                              RES CONSTANT INT-PORT
PRE VANIANLE CHANNEL
REF VARIABLE EMASK2
045 Variable ALV2-6
PEF Variable IVAL
   i) --,
   14
SCH 8 52
    A ( Music - Experimental Constructs

1

2 ( Music board access Judinitions,
                                                                                                              #52481-MPW )
    3 : REGE REG-PORT CHANNEL # + PI ;
4 : DATA DAT-PORT CHANNEL # + PI ;
                                                                                                      ( HEGS ---- )
( HEG-DATA ---- )
    5
5
6 [ fone enable definitions
7 : ENABLE] 7 HEG-PONT P1 D
    7 : ENABLE: 7 NEG-PONT PI DAT-FURT PI ;
8 : ENABLE: 7 NEG-PONT 2 + PI DAT-FURT 2 + PI ;
                                                                                                           I EMASKI -
                                                                                                          ( EMASK2 ---- )
  9

10 (Set nusic loard registers definitions } )

11 : Set ?Terminal if by Emable! By Emable? By Emask1 !

12 : RFF Emask2 ! Aront Them HEGE DATA ; { HEGE HEG-DATA --- }

13 : TON-PEH # VOICE # 2 * + SET ;

14 1 VOICE # 2 * + SET ; { COARSE PINE --- }

15 : AMP # VOICE # + SET ; --> { AMPLITUDE --- }
SCR # 53
d ( Music - Experimental Constructs
                                                                                                              #52483-MPh )
     7 ( Select voice and set enable mask
    1 Sever voice and met enable mask

3 sou (HUILDS C, C, C, DOES DUP 1+ DUP 1+ CW EMASKI # AND

2 WASKI I CM VOICL I CM CHANNEL I ;

5 WFB 2 MAVI VI #FD 1 A BVI V2 #FE 8 MAVI V3

5 sou (HUILDS C, C, C, DOES DUP 1+ DUP 1+ CM EMASK2 # AND

2 WASK2 I CM VOICL I CM COMMNEL I ;

8 WFB 2 MAV2 V4 #FD 1 2 SV2 V5 #FE 8 2 MV2 V4
  9

9 (Heset voice bit in enable mask

11 : vdl (WULLDS C, DOES) CR EMASKI Q DH EMASKI I ;

12 4 vdl vldis 2 vdl vldis l vdl vldis

13 : vd2 (SUILDS C, DOES) CR EMASK2 Q DH EMASK2 I ;

14 4 vd2 vddis 2 vd2 v5dis l vd2 v6dis --->
                                                                                                                                   )
```

MUSIC GENERATION IN FORTH

by Michael Burton

The General Instruments programmable 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 programmable 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 Screen 51 consists of definiscreens. tions 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 I/O 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 set for that voice. Voices are actually played only when a note duration is selected. The definitions vldis through v6dis 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 to be played.

Screens 54 and 55 define the musical notes. The lowest note that may be played is 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 all the registers on each PSG. RINIT is the only place where the amplitude of the voices is initialized, and should be used before playing any music. The definitions VON and VOFF are used to turn all selected voices on and off. They are used inside the note duration definitions and are not meant to be used in a song definition.

Screen 57 contains definitions for rest durations, from a sixty-fourth rest (fr) to a whole rest (wr). It also contains definitions for slur note durations, from a sixty-fourth slur (fs) to a whole slur ws). A slur note is one that does not go if after its duration is finished, allowing a smooth transition between notes when desired. Screen 57 also contains definitions for dotted slur durations, from a sixty-fourth dotted slur (fds) to a whole dotted slur (wds).

Screen 58 is the last of the music constructs screens, and contains definitions for note durations, from a sixtyfourth note (f) to a whole note (w). Note that 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 a 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.

5CK (sic - Experimental Constructs #52681-#PB)
,	te definitions)
- 3 .	(BUILDS C. C. DOES) DUP 1+ CP SMAP CO TON-PEH :	-
	AD2 n b# AE REA n cl AE ALB n c81 AD \$40 n 41	
5	APE n del PR BDA n el AR 82E n fl BA APE n fel	
6	(F7 n q) A9 C58 n q\$1 AB AE1 n a1 AA 061 n a81	
2	0E9 n b] 67 877 n c2 87 88C n c82 86 6A7 n d2 847 n d82 85 8ED n e2 85 898 n £2 85 847 n f82	
8 9	947 n d82 P5 BED n e2 85 898 n f2 85 847 n f82 PFC n g2 84 884 n g82 84 876 n e7 84 831 n e82	
30	PFC n q2	
11	1 224 n d#3 #2 #F6 n e3 #2 #CC n f3 #2 #A4 n f#3	
12	#7Eng3 #2 65Ang#3 #2 638 na3 #2 918 na#3	
15	WFA n b3 #1 #DE n c4 #1 PC3 h c#4 #1 #AA n d4	
14	992 n d94 81 878 n e4 81 866 n f4 91 852 n f84	
15	. #31 n g4 #1 #20 n g14 #1 #1C n a4 #1 ##C n a44	.,
• •		
SCH) us[c - £xperimenta] Constructs 05268]~APr	
	Iste - Experimental Consciucts	• •
	ote definitions)
j	#ED n b4 ## #EF n c5 ## #E1 n c#5 ## #D5 n d5	
	F #C9 n d#5 ## ##E n e5 ## #83 n f5 ## #A9 n f#5	
5	1 #9F n g5 88 #96 n g85 88 88E n #5 88 #86 n #85	
6	8 87P n b5 08 877 n c6 88-873 n c86 88 86A n d6	
7	8 864 n d86 88 85F n e6 88 859 n £6 88 854 n £86	
	8 856 n q4 88 948 n g86 88 847 n a5 88 843 n a86 a 837 n b5 86 830 n c7 88 838 n c87 88 835 n d7	
9	8 832 n 64 86 83C n c7 88 838 n c87 88 835 n d7 8 832 n d87 88 825 n e7 88 820 n f7 88 82A n f87	
10	# #32 n d#7 ## #26 n e7 ## #20 n £7 ## #24 n tu7 # #28 n q7 ## #25 n q#7 ## #24 n a7 ## #22 n a#7	
11	8 #28 n o.7 ## #25 n g#7 #8 #24 n a7 #8 #22 n a#7 8 #29 n b7 #8 #15 n c8 #8 #10 n c88 #8 #18 n d8	
- 15	# #19 n d## ## #18 n e8 0# #15 n f8 #15 n f88	
14	P #14 n q8 #8 #13 n q88 ## #12 n a# ## #11 n #18	
15	8 #10 n hA>	
123456789	USIC - Experimental Constructs 852481-APM lear all music board regs and set amplitudes INIT BFF EMABLE1 0FF EMABLE2 3 0 DJ I CHANNEL 1 7 0 DO 0 I SET LOOP 00 0 DO ALV2-6 1 SET LOOP E 00 D0 0 I SET LOOP 2 +LOOP 0 CHANNEL 1 2 VOICE 1 LV1 0 AMP ; oices enable/disable definitions ON EMASK1 0 EMABLE1 EMASK2 0 EMABLE2 ; OFF 0FF EMABLE1 0FF EMABLE2 ;	9))
ok.		
əci: U 1	/ usic - Experimental Constructs 052481-AP	·•)
;	est duration definitions)
5	(BUILDS C, DOES) CP DVAL # SWAP / # DO LOOP ;	
	4 r fr 32 r tr 16 r sr #8 r wr	
5	4 rgr 82 rhr 91 rwr	
6		
7	lur duration definitions)
8	I CHUILDS C, DURSS CA DVAL & SWAP / & VON DO LOOP ; 4 sl fs 37 sl ts 16 sl ss 48 sl es	
9	4 s] fs 37 s] ts 16 s] ss 48 s] es	
14	4 sì qs #2 sì hs #1 sì ws	
- 11		
	otted slur duration definitions)
13	AS DVAL # 54 / DUP 2 / + # VON DO LOOP ;	
	ds ts fs ; ; sds ss ts ; ; eds es ss ; de ds es ss ; ; sds ss ts ; ; eds es ss ;	
15	yds qaes; thdshsqs; twdswshs; ~->	

```
SCR 0 58
0 ( Music Experimental Constructs
                                                                                                                      #52481-### 3
        ( Note duration definitions
        ( NOCE GUIALLOS GETINITIONS

: d (SUILDS C, DUES) CA DVAL A SMAP / 8 VAN DU LUOP

VOPF IVAL 0 B DU LOOP;

64 d f 32 d t 66 d 66 d e

94 d g 92 d h 81 d w
  7

8 (Dotted note duration definitions

9 : do VOPF IVAL (# 18 DO LOOF;

19 : fd fds do; : td tds Jo; : sd sds du;

11 : ed eds do; : tqd uds do; : hd hds do;

12 : vd wds do;
                                                                                                                                              ,
   12 : WO WOB GO ]
13 DECIMAL HINIT
14 CP ." Music Constructs Loaded "
15 :5
SCR # 59
6 (
1
2
                                                                                                                      #52481-428 )
   12
   13
   15
....
 SCH 8 64
P ' Song - Ned Hiver Valley

Jony - Net Hiver Valley
B51201-
HHI V4 C5 us f5 e v2 a5 v3 f5 v4 C5 v5 f3 es c4 e
v3 a5 v5 d84 v6 d3 es v2 c6 v3 f85 v5 de v6 d3 e v2 a5
v3 f5 v4 b4 v6 q3 q v2 d5 v3 e5 v4 a84 v5 c4 v61s q
v2 f5 v3 c5 v4 f4 v5 c4 v6 f3 qs v2 a5 v3 f5 v4 c5 es
v2 f5 v3 c5 v4 f4 v5 c4 v6 f3 qs v2 a5 v3 f5 v4 c5 es
v2 c5 v3 c5 v4 f4 v5 c4 v6 f3 qs v2 a5 v3 f5 v4 c5 es
v3 a5 v2 a5 v3 c5 v4 f4 v5 c4 v6 f3 qs v2 a5 v3 f5 v4 c5 es
v1 a5 v2 a5 v3 c5 v4 f3 v5 f3 es v4 c4 e v4 e4 es
v1 a5 v2 a5 v3 a64 v4 c4 ev 1 f5 v2 f5 v3 a64 ef f4
v5 f3 q v1 q5 v2 q5 v3 d5 v4 e4 v5 c4 v6 f3 es v1 f5
v1 c5 v2 f5 v3 d5 es v4 g4 v5 a83 es v4 e6 es d4 q i
v5 f5 v1 b5 v2 f5 v3 d5 es v4 g4 v5 a83 es v4 e6 es d4 q i
v3 f5 v2 f5 v3 d5 es v4 g4 v5 a83 es v4 e6 es d4 q i

                                                                                                                       #512#1-#Pb )
    13
14
   15
 SCR # 45

Ø ' Song - Red River Valley
                                                                                                                         #51281-MPH )
      13
    15 --->
  SCH # 66
      # ( Song - Hed River Valley
                                                                                                                         851283-6PH 1
    13 -->
    14
  SCH # 67
                                                                                                                         #52481-MP# )
       # [ Song - Hed River Valley
      RRJ RR2 RR3 NH4 RR5 RR2 RN3 HR4 RR5
RR2 RR3 RR4 RR5 RR7 RINIT ;
CR ." Red River Valley loaded "
     11
12
13
14
15
           :5
   ....
```

#53.783-486.3 SCR 0 70 0 (Song - Jesus Christ Superstar 1 #51281-MPH) b) (administry) (additional additional ad 14 SCH # 71 # { Song - Jesus Christ Superstar \$51281-MPa) 1 . JC6 v2 a5 v3 d05 v4 f3 v1 c6 e ss v4dis s v4 c4 e t4 2 : JC6 v2 a5 v3 d05 v4 f3 v1 c6 e ss v4dis s v4 c4 e t4 4 a05 v2 t5 v3 d5 v4 g3 e v1 c6 e d66 v2 a05 v3 e5 v4 c4 5 v1 c6 v2 g5 v3 e5 es v4 g3 q v1 dis v2 d05 v3 d5 s 6 : JC7 c4 es v1 g5 s a 465 v2 f5 v3 d5 s e5 v1 c6 v2 g5 e 7 ss h v1 d1s v2 d1s v3 d1s v4 f4 es v1 g5 s s a05 v2 g5 8 v3 d05 s v1 c6 v2 a5 s s v2 f5 s v2 d1s v2 d1s v2 d1s v3 d1s 9 f4 es v1 a5 s s v2 f5 s v2 d1s v1 c6 e f6 v2 c6 v3 a5 11 \rightarrow 12 18 11 17 13 14 15 SCH 6 72 8 (bong - Jusus Christ Superstar 1 $\begin{array}{c} \text{sound} = \text{sense christ Superstar}\\ 1\\1\\1\\1 \text{ ; JCE vldis v2dis v3dis v4 c4 es v1 ef s s f6 v2 c6 v3 e5 s 3\\2 v2dis v3dis v1 d66 e c6 v2 g5 v3 e5 ss qs v4 g3 es e 4\\2 v1 dis v2dis v3dis c4 es v1 g5 s s a63 v2 f5 v3 d5 s e5 s v1 c6 v2 g5 e ss qs v4 g3 es c4 e vldis v2dis v3dis c5 s v1 c6 v2 g5 e ss qs v4 g3 es c4 e vldis v2dis v3dis c6 f4 es v1 a5 s s v2 f5 v3 d5 s e5 v1 c6 e f3 es v1 a5 s s v2 f5 v3 d5 s e5 s v1 c6 e f3 v2 c6 v3 a5 ss v4 c4 qs f3 es c4 e i d1 s v2dis v3dis v3 d1 s c4 e i 3 es c4 e i d1 s v2dis v3dis v3 d1 s c4 e i 3 es c4 e i 1 es v1 f6 s f6 v2 c6 v3 a5 ss v4 d64 le qs c4 es f3 e v1dis v2dis v3dis v3dis c4 es v1 f6 s s f6 v2 c6 v3 a5 ss v4 d64 le qs c4 es f3 e v1dis v3dis v1 d66 e c6 v2 q5 v3 e5 ss v4 a63 qs q5 q; l3 -> l4 l5 v3dis v3dis v3dis v3 d66 la v3 v3 v3 v3 v3 v3 v3 e5 ss v3 v3 e5 ss v3 e3 v3 e$ #51581-MPH) SCR 0 73 Ø (Song - Jeaus Christ Superstar 1 #51281-MPH > 1 2 : JCl0 vl g5 v2 a5 v3 c5 v4 g4 v5 c4 v6 c3 q v2dis v3dis v4dis vl e5 e c5 v2 q4 v3 e4 esh vl a5 v2 f5 v3 c5 v4 a4 v5 f4 v6 f3 q v2dis v3dis v4dis vl f5 e c5 v2 a4 esh f5 vl a85 5 v3 d5 a84 v5 a83 v6 a82 q v2dis v3dis v4dis vl g5 e a85 e a5 v2 f5 v3 c5 v4 a4 v5 f4 v6 f3 q v2dis v3dis v1 dis v1 g5 7 e f5 e g5 v2 e5 v3 c5 v4 g4 v5 v5 c4 v6 c3 q v2dis v3dis v4dis vl e5 e c5 v2 g4 v3 e4 es es v6dis v5 g3 qs e v5dis ; ---> ---» 18 11 12 13 14 15 SCR # 74 # (Song - Jesus Christ Superstar 65248)-MPB)

.

OPTIMIZING DICTIONARY SEARCHES

Paul van der Eijk 5480 Wisconsin Avenue, #1128 Chevy Chase, MD 20015 (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 times 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 When (FIND) searches the of the name. dictionary for an entry, the lengths of the strings are compared. If the comparison fails, and this happens a lot, the characters stored are scanned for a high bit in the last character. When the scan stops at the last character, we know the address of the LFA, because it follows the It will be obvious that last character. 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 Another approach was taken by stored. Robert Smith, see FORTH DIMENSIONS. Vol. 1, 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 LFA 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.

0 (LFA preceeds NFA 1 of 2: 2 : CREATE -FIND IF DROP NFA 1D. 4 MESSAGE SPACE THEN 3 (check unique in CURRENT and CONTEXT) 4 HERE CQ WIDTH Q MIN >R (save number of chars stored) 5 HERE OAD TOGCLE HERE R + DUP OBO TOGCLE 6 (smudge and delimiter bits) DUP 2+ R 1+ -CHOVE (move entry down to insert LFA) LATEST HERE 1 HERE 2+ CURRENT \emptyset 1 R> 3 + ALLOT HERE 2+ , ; 10 11 : NFA 3 - -1 TRAVERSE ; 12 : LFA NFA 2 - ; : PFA 1 TRAVERSE 3 + ; 13 14 15 ---0 { LFA preceden NFA 2 of 2: Paul van der Eij 1 : VOCABULARY <BUILUS CURRENT @ 2+, OAO81 2 HERE VOC-LINK @ , VOC-LINK i 3 DOES> CONTEXT i ; Paul van der Eijk april-12-1981) { the following change in -FIND speeds up dictionary searches)
(ip case the CURRENT and CONTEXT vocabularies are the same.)
(the change is not necessary for the new dictionary structure) 8 : -FIND BL WORD HERE CONTEXT & & (FIND) DUP 0-If drop latest context & & (Ver -If here Smap (Find) Else drop 0 10 11 12 13 THEN THEN ; 14 15 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.

12

,

TRACING COLON-DEFINITIONS

Paul van der Eijk 5480 Wisconsin Avenue, #1128 Chevy Chase, MD 20015 (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.

÷e

vi is

72

-1

75

fa

ed.

Ξ.

S-c

20

57

`>e

28

0t

fo

ti

Po

Gr

í 💼

C

Se

æ

PRI

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 inserted.

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

i.e.:

O TRACE ! : TEST1 T11 T12 ; (TEST1 will not be traced)

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

U (trace culon words: Paul van der Eijk april-12-1981) 1 PORTH DEFINITIONS 2 O VARIABLE TFLAC (controls trace output) 3 : (TRACE) (give trace output. to be inserted as first word) 4 TFLAG @ (trace output if non-zero) 5 IF CR R 2 - NFA DUP ID. (back to FFA NFA for name) 6 C@ 31 AND 32 SWAP - SPACES (add spaces to name) 7 -2 4 DO SP@ 1 + @ 8 .R -2 +LOOP (show stack) 8 THEN ; 9 : (redefined to insert trace word after colon) 10 ?EXEC ICSP CUREENT @ CONTEXT i GREATE 11 ' (TRACE) CFA DUP @ HERE 2 - 1 .] ; IMPEDIATE 12 13 (example: trace following use of 1 and C!) 14 : 1 1; 15 : C1 C1 ; ;5

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

e're Growing

FORTH, Inc. expects to double its staff within the next year to accommodate increased product demand and applications programming. (See job openings listed where in this publication.)

The latest addition to our staff is programmer Charles Curley. Curley is a former freelance writer and programmer who edits and publishes the <u>Ohio Scientific</u> <u>sers' Newsletter</u>.

"I put FIG FORTH up in my own Ohio Scientific C2-8P DF and liked it," he comments, "but I wanted to learn FORTH systematically, and I figured this was the test place to do that. At FORTH, I get taid to do what I like to do."

Ither News

President Elizabeth D. Rather was a member of a panel on programming languages for small computers at the NCC Convention. She was featured in both <u>Computer-</u> world and Computer Business News.

Programmer Mike LaManna has relocated to Long Island, New York, and is working to the 68000 polyFORTH. It should be available midsummer.

FelyFORTH Palo Alto Users Groups Starting

Dr. C. H. Ting has volunteered to Chair the Palo Alto Thread of the FORTH Users Froup for the first three months. Anyone interested in joining the Users Group may contact Dr. Ting at Lockheed Missiles and Space Corp., (408) 742-1101 or Al Krever at FORTH, Inc. (213) 372-8493.

Fecent Applications

FORTH, Inc. has produced a computer tumerical 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 yield 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 <u>Starting</u> 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

Seminar	Workshop	
August 4	August 5	
August 6	August 7	
September 1	September 4	
October 15	October 16	
October 22	October 23	
	August 4 August 6 September 1 October 15	

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

FORTH VENDORS

The following vendors have versions of FORTH available or are consultants:

ALPHA MICRO

Professional Management Services 724 Arastradero Rd. #109 Palo Alto, CA 94306 (415) 858-2218

Sierra Computer Co. 617 Mark NE Albuquerque, NM 87123

APPLE

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) 321-4340

CROSS COMPILERS

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

polyFORTd

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

LYNX

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

M & B Design 820 Sweetbay Drive Sunnyvale, CA 94086

Micropoli.

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

North Star

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

05I

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

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

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

Tom Zimmer 292 Falcato Dr. Milpitas, CA 95035

6800 6 6809

Kenyon Microsystems 3350 Walnut Blvd. Houston, TX 77042 (713) 978-6933

PDP-11

Laboratory Software Systems, Inc. 3634 Mandeville Canyon Rd. Los Angeles, CA 90049 (213) 472-6995

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

TRS-80

Miller Microcomputer Services 61 Lake Shore Rd. Natick, MA 01760 (617) 653-6136

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

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

KDI

Eric C. Rehnke 540 S. Ranch View Circle #61 Ansheim Hills, CA 92087

8080/280/CP/H

Laboratory Microsystems 4147 Beethoven St. Los Angeles, CA 90066 (213) 390-9292

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

Consultant

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 Firmware, Boards and Machines 10

Ť

ť

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

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

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

Consultants Creative Solutions, Inc. 4801 Randolph Rd. Rockville, MD 20852

Dave Boulton 581 Oakridge Dr. Redwood City, CA 94062 (415) 368-3257

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

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

Inner Access P. O. Box 888 Belmont, CA 94002 (415) 591-8295

DECOMPILER FOR SYN-FORTH

SYN-1 User's Group PO Box 315 Chico, CA 95927

The following decompiler works very well except that because INTERPRET is not remembered by ;S nor (;CODE-) nor QUIT, this FORTH decompiles loop !

```
UNI 2 LIST J LIST
 50801
     O ( DECOMPILER... ) HEX
     2 VOCABULARY UTIL INNEDIATE PORTH DEFINITIONS
       - PICK 2 * SPR + R ; UTTL DEFINITIONS
     + : TEST- ( PFA P PFA? -> FFA P ) 3 PICE = OR ;
     P : TEST.FND ( PCFA -> PCFA P ) DHP P 2+ D
               ';$ TEST*
'(;CODE) TEST*
'QUIT TEST*
   10
                            TTERMINAL OF SWAP DRUP (** :
   12
   :4
15 --->
  SC8 / 2
     0 ( ... BECOMPILER... )
       INCEDENT ( CCTA -> CCTA+ ) DUP 3* SUAP @ 2+
0 'CONTILL TIST- IF DROP DUP @ 2+ NFA 10. 2+ SFACE
ELSE 0 '(...) TEST- IF DROP DUP C@ HER .> 1+ SFACE
ELSE 0 '(...) TEST- IF DROP DUP COUNT DUP AND
TYTE+ 1+ 22 DNIT SPACE
ELSE 0 'LIT TEST- UP DUP IF NEE THEM
'OBAINCH TEST-
'(OBAINCH TEST-
'(COUP) TEST- IF DROP DUP @ . SFACE 2+ ELSE DROP
THEM THEM DECEMAL THEM ;
   10
11
12
    15 -->
 SCR # 3
0 ( ... DECOMPILER... ) FORTH DEFINITIONS HER
      DECONTILE ( PPA -> NFA OF HERT WORD ) UTIL
HEX DUP 4 .E DUP CFA @ 5 .R DECIMAL
DUP CFA @ 803 =
IF ... DUP HFA ID. SPACE DUP
BECIN DUP @ 2+ HEA DUP CØ 40 AND IP ." [COMPILE] " THEN
ID. TEST.ED
WHILE INCREDENT
REPEAT
DROF DUP HFA @ 40 AND IF ." INHEDIATE " THEN
ELSE ... " DUP HFA ID.
THEN LFA @ CR CR ;
  15
  14 . DECOMPILE.ALL ( PFA -> PFA ) CR
15 BEGIN DECOMPILE DUP PFA SMAP O- TTERMINAL OR UNTIL CR ;
 œ
 EXAMPLES
       DECOMPILE. ALL

    DECOMPLE.ALL
    1A17 803: . 5->0 D.;5
    1A07 803: . 8.>>0 D.;5
    1A07 803: . A.>A 5->0 R> D.;5
    1974 803: D. O.B. SHACZ;5
    1975 803: D. O.B. SHACZ;5
    1950 803: 65 $ OVER OVER ON 0- OBRANCH -12;5
    1951 803: 6 AST & HUTOL ROT CLIT 9 OVER < DBRANCH 7 CLIT 7 + CLIT 30 + B</li>

IAFI AO3 : IF CENFILE DBRANCH HERE 0 , 2 ;5 1997EDIATE
1808 - 803 : REFEAT >8 "R [CONFILE] AGAIN-R> R> 2 - [CONFILE] ENDIF 1998EDIATE
   WILE DECOMPILE 1924 803 : WILE [COMPLE] 17 24 15 DIMEDIATE
```

4 OK

i, Im

72

```
0 COLON DEFINITIONS DECOMPILES FOR SYN-1

1 MICHEL DESSAINTES MARCH 22, 1961

2 USE : 'WORD.TO.DECOMPILE BECOMPILE

0 BI : 'PIEST.WORD.TO.DECOM DECOMPILE.ALL

5 OUTPUT : PFA CFA.COWTENT : WORD.TO.DECOMP ....

6 OUTPUT : PFA CFA.COWTENT : WORD.TO.DECOMP ....

7 OTHER LITTERAL ARE IN DEFINAL (BRANCH, OBRANCH, LIT,...)

9 OTHER LITTERAL ARE IN MEXADECINAL (LIT, CLIT,...)

10 POSSIBLE EXTENSIONS : REPLACE (LOOP), (BRANCH),....

11 DOFSIBLE EXTENSIONS : REPLACE (LOOP), (BRANCH),....

12 BT LOOP, BRANCH,....

13 JUSTIFI 'P .... ELSE .... ELSE, ....

14 BECOMPILE VARIABLE, VOCABULARY, ....

15
```

SCR # 4

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 32K of dynamic RAM as storage space 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 OX6 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
- 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

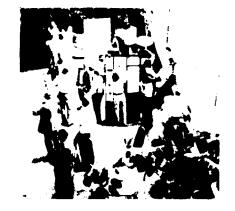
MEETINGS/EVENTS ANNOUNCEMENT FORMAT

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. REMARKS 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)

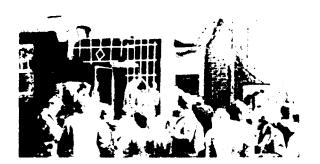
FORTH STANDARDS CONFERENCE ROCHESTER, NY - SPRING 1981











FORTH DIMENSIONS 111/2

Ŀ

, a g

,

٤,

:e 1g 1] 1)

ea rk ed

12

1981 FORML CONFERENCE

Asilomar, California November 25-27, 1981

ATTENDEE REGISTRATION FORM

ConferenceThe 1981 FORML (FORTH Modification Laboratory) is an advanced seminarPurpose:for the presentation of FORTH language papers and discussions. It isnotintended for beginning or casual FORTH programmers.

AttendeeThe FORML Conference is limited to 60 FORTH programmers (approx. 30Selectionfamily and other non-participants accommodations are also available).Priority: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.

Registration Form, Complete and return to: FORML PO Box 51351 Palo Alto, CA 94303	NAME
	ADDRESS
	CITYSTATEZIPCOUNTRY
	PHONE (Day)(Evening)
	I have been programming in FORTH for: (years) (months)
	Types of CPU's and/or computers:
	I have authored the following papers and/or articles about FORTH:
	I expect to: present a paper, present a poster session chair a section, non-presentor
	My topic will be:
Accommodations Desired:	Rooms at Asilomar include meals (including a huge Thanksgiving) and the price of the Proceedings is included in participant costs.
	Myself Double \$110.00 Single \$150.00
	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 \$47.00 single
	FOR MORE INFORMATION CALL: ROY MARTENS (415) 962-8653

FORTH DIMENSIONS 111/2

LATE NEWS

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

June 29, 1981

Mr. Roy C. Martens 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 currently available on the market, namely the Laboratory Microsystems, Z-80 FORTH.

I bought Laboratory Microsystems Z-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 4MHz for my and 6Mhz for Timins systems on which the Laboratory Microsystems Z-80 versions were compared (how convenient is was tried on Mr. Timins systems) the Z-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 <u>very large</u> apology to LABORATORY MICROSYSTEMS, and <u>at least</u> 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!

Sincerely, Glenn A. Burklund

Fublisher's Comment: The following letter was received in feference to a Product Review by C. H. Ting in FORTH DINENSIONS, III/1, page 11-12, which compared some bench marks 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 any unintentional damage to Laboratory Microsystems; to ask our members whether they iesire comparisions between FORTH and other languages and between FORTH products; if we 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?

L

ī

How to form a FIG Chapter:

- 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 x 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, (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", Aloha, Eric Smith, (503) 642-1234.

New England

Boston

lst Wed FIG Meeting, 7:00 p.m., Mitre Corp., Cafeteria, Bedford, MA. Call Bob Demrow, (617) 389-6400, x198.

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, OK. 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) 387-6976. John Hastings (512) 835-1918.

Mid Atlantic

- Potomac Paul van der Eijk, (703) 354-7443 or Joel Shprentz, (703) 437-9218.
- New York Tom Jung, (212) 746-4062.

Midwest

Detroit Dean Vieau, (313) 493-5105.

Foreign

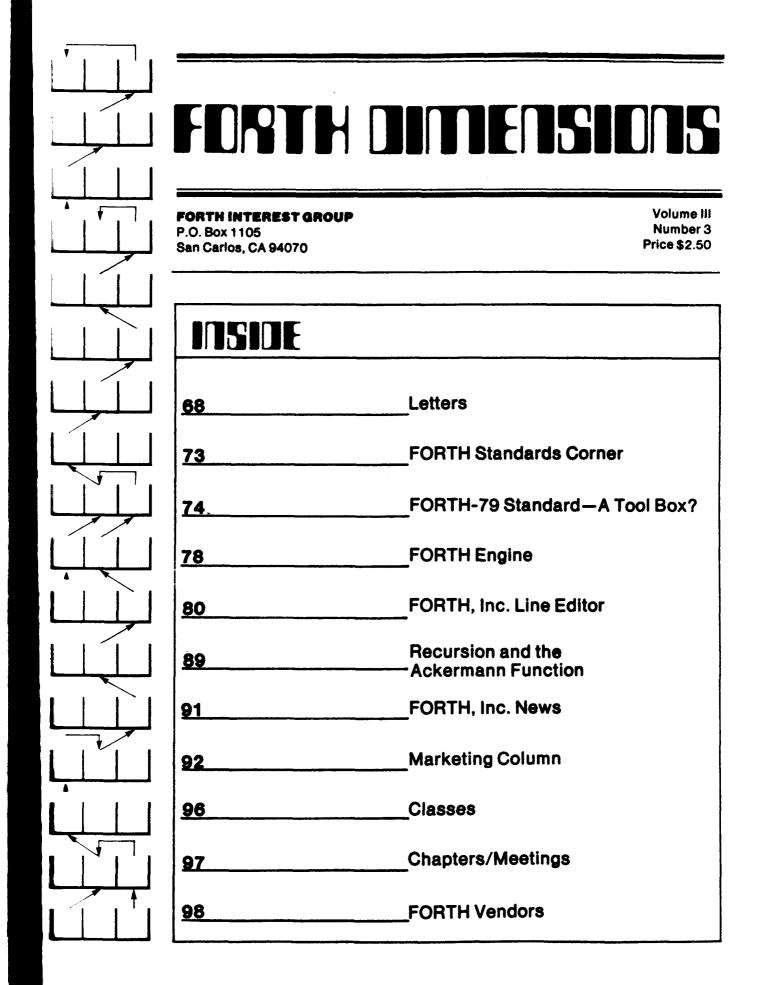
- England FORTH Interest Group, c/o 38, Worsley Road, Frimley, Camberley, Surrey, GU16 5AU, England
- Japan FORTH Interest Group, Baba-bldg. 8F, 3-23-8, Nishi-Shimbashi, Minato-ku, Toyko, 105 Japan.

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. 1



72

-2 3)

, -

•

,

Ø

FORTH DATIENSIONS

Published by Forth Interest Group

Volume III No. 3

September/October 1981

Publisher 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 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 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

HISTORICAL PERSPECTIVE

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

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

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

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 will be devoted to music, graphics and games. Currently, this type of application is in very short supply and I am appealing to our members to submit them as soon as possible. Please remember, YOU DON'T HAVE TO BE A WRITER—our staff 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.

Please 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, Anaheim, 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 fiyers, 8½ x 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' 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 available. 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

LETTERS

in Fig:

6

L

►

•

>

d

Ή

١e

70

in

ity

10

10-

to

A

18-

he

nd

les

the

mo

.

il to

'oxiwe'll

jex-

Mar-

only

185 8

1982

42C:

sland

cur-

sther.

(415)

same

łe

is m disturbed to see that you have recently mismed a review comparing our Z-80 FORTH mage to TIMINS FORTH and stating that the FORTH is significantly slower than the wersion. Please be informed that the mark was run on an early version of Z-80 which has not been distributed for the ming has been benchmarked by some of mistomers as 5-10% FASTER than 8080

he have made an effort to provide a high mit., comprehensive FORTH program develment package at a very reasonable cost. To copies of the current Z-80 FORTH mann have been sold so far and I do not know for unsatisfied customers. I think it reflects to poorly on your publication that you would must such a review without checking the facts all of the interested parties. If you people really concerned with promoting the such attacking the reputation and products for FORTH language vendors.

> Ray Duncan LABORATORY MICROSYSTEMS 4147 Beethoven St. Los Angeles, CA 90066

We fail to see how publishing a comparison where of products that were on the market less where a year ago can be construed as an attack weither that product or the reputation of its proor.

No inference has been made that Laboratory icrosystems Z-80 FORTH is an inferior rouct, or that it has dissatified customers. In received, has any indication, quite the contrary rouct which versions were being comered, but we know of no facts that were inacurately reported. If we are to provide the CRTH world with useful product reviews, rouracy is important, and if Laboratory forosystems knows of any inaccurate facts or rould care to provide us with an updated evew, ample space will be provided.--ed.

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 1630 Worcester Rd., #630C 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 Rancho Bernardo 12615 Higa Place 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? I'm thinking about writing something for Nov./Dec. GRAPHIC/MUSIC. I have implemented a set of graphic words for Columbia Data Products' MX-964 (Z-80 Micro-*2, 512x256 bit mapped, \$10-8080-figFORTH), and am working on some musical words for a dual GI'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 Prentice-Hall 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 Public Relations FORTH, Inc. 2309 Pacific Coast Highway 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 S100 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 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.

Keep up the good work with the magazine.

Bill Miles PO Box 225 Red Cliffs Victoria 3496 Australia

Thanks for your comments, Bill. Glad to The FORTH is alive and well in the land down inter! How about some of our members intresponding with Bill and helping him over the Thigh spots.--ed.

Iter Fig:

٦

e

e

d

е

٧

đ

C

d

-

'S

:0

15

T

G

:0

١ġ

is.

2"

าต่

nt

îx re

SY

he

en

۱6

iШ

ve

er

tc.

ΞX

of

FOR TH DIMENSIONS has grown increasingly zeful to me in the past few months as I have Traily begun to "get the hang of" FORTH. I The running on my TRS-80 several versions of E-SIC, FORTRAN, PASCAL, APL, SAM76 and $- \mathcal{P}$ under both TRSDOS and CDOS: but I have mere: found any language harder to learn that TIFITH. Part of the problem is the scope of at the same time I'm trying to meerstand the interpreter, compiler, OS, and a metax as difficult as LISP's. I have found all me instruction manuals so far to make a drastic from simple concepts like 2.2 + . and IC...LOOPS to discussions of the Dictionary st of Defining Words. (I think the writers had same problems I have, of separating the war ous functions of the system.) One of the helps I received was Mr. Bumgarner's Stack Dacram in this year's March issue of BYTE. The necessity of being able to visualize the sec< cannot be overemphasized. Once I was to do that, I starting learning in earnest.

-aving leaped this hurdle, I found FORTH revearding than any other language to plications programmer is its (almost) reveared of its greatest advantages to me as pplications programmer is its (almost) reveared of the stack of the stack reveared of the stack of the stack of the stack of the stack reveared of the stack of the stack. All reveared of the stack of the stack of the stack of the stack. The stack of the stack. The stack of the stack. The stack of the stack. The stace their return values on the stack. All reveared of the stack of the sta 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 Their product is excellent: was worthwhile. 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 programs. A++ for demonstrations 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 1A8

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

1/3

Dear Fig:

I very much enjoyed my first pass through your article "Compiler Security" in FORTH DIMENSIONS 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 so, 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 timeerror-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 Sr. Principal Programmer Analyst NCR CORPORATION 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 either 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.

tr

5

10

Ы

Ξı

1

fa

th

63

97

-

at i

fier,

Ch

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 END-CODE, 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 SHAW LABS LTD. PO Box 303 San Lorenzo, CA 94580



FIG Convention Coming -- Nov. 28

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

Often in creating a definition, we want to test or output an ASCII character, using words ...ke EMIT or = or possibly even in a CASE statement. These are normally supplied as ...teral numbers in the current radix. These compile into the usual dictionary pair of LIT followed by the literal value. The difficulty is that 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 editoracceptable character to be displayed in its real form while compiling into a normal literal pair. While this may prove to be a minor help at edit-time, the resultant source code is much more readable at a later time, and is selfcommenting, both highly desired features of any programming language.

The new word is ASCII, and it is followed by a literal character. The definition of ASCII is ample:

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

It is made immediate so that it executes during compile-time. WORD takes the next inputstream text, delimited by a blank, and places it at HERE. Then the first character is placed on the stack for use by LITERAL, which has been forced to be compiled into our definition. What could be easier?

Formerly, we had to write 65 EMIT to outbut the letter "A" (assuming decimal radix). "Now we can write ASCII A EMIT, clearly the tetter for everybody's understanding. The thouse of the word ASCII is open to change, but the idea is a valuable addition to our efficient use of the language.

That's my contribution. I hope others can use it to improve their work. Thank you for another a medium for ideas.

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

MULTIPLE WHILE SOLUTION

I 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) WHILE. . . 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 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

3

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 181

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 stored 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

301

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 the input stream is exhausted when WORD is called.

The problem of the character count limitation could be considered in the future. One imple approach would be to use a full word for the character count. Another would be to elimnate the character count and always append a rull at the end. The user could then do his own scanning. The problem of null length strings rould be "defined" away by making null length strings illegal. I think that that is a poor sclution. The real problem is that WORD is roorly factored. As usual in FORTH, the less a word does, the more useful it becomes. The process of scanning for initial delimiters should re separated from the process of scanning for terminating delimiters.

NEW PRODUCT

Z8002 Software Development System under $\mathbb{CP}/M^{\textcircled{0}}$ or Cromemco CDOS. Includes cross compiler and a number of utilities. Available on $5\frac{1}{4}$ " or 8" 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 xxx, 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 standard 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:

2

2

Ł

ĩ

t

*

oc fc

đ

t

50

e

t

9

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 <u>only</u> 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 system 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 Still not clear? Send in the standard. We need them to make a better auestions. 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 Shaw Laboratories, Ltd. P. O. Box 303 San Lorenzo, CA 94580

Book Review:

I

1

Ł

٦

٦

١,

е

d

f

n

e

S

13

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

In most books the useful information begins .n chapter one, or later. <u>Starting FORTH</u> is an exception. Useful information starts in the Forward section of the book.

The book is designed to be interactive. After only two pages of chapter one, you are typing at the terminal. It is seldom that a sentence will leave you thinking, "Now, what does that mean?" Analogies are used througnout. Any "buzz" words, or differences between systems, or phrases which might cause confusion are footnoted to explain in more detail. This presents extremely basic or nonessential Jeas outside of the main text, allowing the book to be simple enough for the beginner, but not to become tiresome to the more knowledgeable. For example, in the sections of the book cealing with math, separate sections or footnotes are presented to explain what integers are x what an absolute value is (for beginners). Or to give additional information about a faster algorithm than was used in an example in the main text (for experts). Where appropriate, guizzes or exercises are interspersed within the chapters to help with understanding the material presented.

The book is written for the current "closeto-79-STANDARD" version of polyFORTH with notations or footnotes to indicate and explain the differences from the standard. Throughout the book, tables and lists are used to summarize and clarify the information presented. The occasional tables of new words (in glossary form) are of great help. They prevent having to dig through the text for the words to perform At the end of each the practice problems. chapter is a complete glossary of the new Also, at the end of each chapter are words. problems, 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 LISTing, 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 also 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 buffer 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 immediate 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 <u>Starting Forth</u>.

On the whole, <u>Starting Forth</u> 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 conventional Von Neumann computers is useful but dull. This in spite of large vendors' advertising literture which breathlessly announces new architectural advances for their latest machines. Meyers' book¹ has an entertaining discussion of the history of these "new" advances. For example, virtual storage goes back to the Atlas system (U. Manchester, 1959).

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 programmers 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 going out of range is a common run time error but 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 language. We can do this at several levels, the owest being language-directed design where hardware features are added to support specific An example would be language features. Burroughs' concept of data descriptors to provide run time checking of subscript ranges. Another example would be a P-code machine. P-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/1 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/1-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.

L

- Postulate hardware data paths, stacks, registers, etc., that implement DO and mating primitives such as LOOP and
- Make sure this hardware supports c. hidden logical concepts -- in this case, I, J, K-and violates no other FORTH concepts.
- d. Count clock cycles.

+LOOP.

ь.

- Repeat b-d until you can think of no e. 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 1-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 result⁴ 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 16k X 16 main memory with 1k X 16 stacks both extendable by 4 X. I/O is done with a slave 6809 vith programmed access to the data stack and DMA access to main memory. Control is microprogrammed with a 2910 driving a 1k X 60 bit writeable control store. This follows Logic Engine philosophy so the user has very pleasant access to the micromemory for high-speed special purpose tailoring instructions.

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

DUP 99 X faster	SWAP	132 X faster
a 101 X faster	U#	96 X faster
! 114 X faster	ROT	624 X faster
AND 126 X faster	DOLOOP	110 X faster
		(null body)

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 Z80. I would appreciate hearing from readers about this as well as memory and I/O requirements.

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

REFERENCES

¹Advances in Computer Architecture, Glenford J. Meyers, Wiley, 1978.

²Southcon Conference, Atlanta, Georgia, 1981, Session 20/2, Phillip Crews.

³SYMBOL - A Major Departure from Classic Software Dominated Von Neumann Computing Systems. Proc. 1971 Spring Joint Computer Conf., AFIPS, 1971, pp. 575-587.

⁴Southcon Conference, Atlanta, Georgia, 1981, Session 20/4, David Winkel

⁵The Art of Digital Design, D. Winkel & F. Prosser, Prentice Hall, 1981.

F 3

a

5

e

đ

ġ

-

Ŀ

t

DI

2

a

21

21

T

la

 $\mathbf{\alpha}$

fu

bu.

S

THE FORTH, INC. LINE EDITOR

S. H. Daniel System Development Corporation 500 Macara Avenue Sunnyvale, CA 94086

The upcoming publication of Starting FORTH, which is destined to become the "bible" of FORTH neophytes everywhere, provides an coportunity to upgrade the existing fig-FORTH -ne 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 -ill be able to step up to a polyFORTH system and use the line editor. Conversely, FORTH, inc. customers who try fig-FORTH will not have to learn to use a different editor.

The second reason for adopting the colyFORTH editor is its increased flexibility and ease of use. The current fig line editor uses only the PAD for storage of user inputs for searches, deletions, and replacements. The colyFORTH editor employs both a FIND buffer and an INSERT buffer, in addition to the PAD. This allows both of the extra buffers to be caded, and the contents reused several times, without extra typing by the user. This makes commands like D (Delete) and R (Replace) especially useful.

By taking a few hints from Starting FORTH, and combining them with the existing editor, I was able to write a line editor which is functionally identical to the polyFORTH editor, but which is in the public domain and can be used by anyone.

SYSTEM REQUIREMENTS

This editor should run on any fig-FORTH sestem, including FORTH-79 Standard systems if the changes mentioned in the section FORTH-79 Standard are made). The compiled re editor requires approximately 2K bytes of remory, plus room in the system for the PAD and the FIND and INSERT buffers. It operates within the confines of the default data and return 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

Т

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.

X eXtract (--)

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.

Type (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 Next (--)

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.

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

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 Find (--) F text

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.

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.

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.

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.

> S Search (last screen#+1 --) S text

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, S 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.

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 Under (--) U text

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.

Any following text is copied into the INSERT buffer. The R command operates as a combination of the E (Erase) and I (Insert) commands. Starting at the current cursor position, and working backwards towards the start of the line, text corresponding to the ł

5

t

5

u

ħ

t

U

Ii.

G

F

TE

length of the contents of the FIND buffer is beleted, and the contents of the INSERT buffer are inserted into the line. Since the contents of the FIND buffer determine how much text will be erased, the R command should only be used immediately following the F (Find) command.

M Move (Block#, Line# --)

Copies the current line into the INSERT buffer, then copies the INSERT buffer to the block, specified by Block#, UNDER the line specified by LINE#. The original block number is testored, and the next line in the block becomes the current line. This allows sequential lines to be moved with a minimum of keystrokes. One unfortunate side-effect of this command is that to move something to line 0 of another screen, ou must first move it UNDER line 0, using the command xxx 0 M, make screen xxx current, and then extract the old line 0, moving everything else up.

(--)

Used as a terminator for all commands allowing text input, such as P, F, R, etc. Allows more than one command to be entered on a single the, e.g.,

3 T P This is line $3^{\uparrow} L$ (cr)

Although useful, this feature does preclude the use of the " " as a character in any text to be put on a screen.

GLOSSARY

ł

;

t

9

þ

•

9

2

8

1

B

đ

9

T

8

:)

ſ

e e

ī

Ť

The following glossary addresses all the FORTH words in the line editor except the actual editing commands, which are discussed above.

'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.

1LINE (-- 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.

(F) (--)

Copies any following text to the FIND buffer and searches the

F

current screen for a match. Used 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.

- COUNTER A variable used by the S command to count the number of lines output to the screen and printer.
- **SUMP** (--)

t

)

P

t e D ρ r

b }.

۱-

e

d

he he

13

Increments the number of lines output and sends a page eject when 56 lines have been output. Used by the S command to handle pagination on the console and printer.

FORTH-79 STANDARD

The following changes should be made to the load 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 5,6,7 The FORTH word R should be changed to R@.

- 212 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 de-202 leted, since the FORTH-79 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.

SCR # 200 816715 SHL > 0 < polyFORTH compatible line editor 2 FORTH DEFINITIONS HEX DAT (accept following text to PRD) HERE C/L 1+ BLANKS WORD HERE PRD C/L 1+ CMOUE \$ 5 : TEXT 6 8 (relative to SCR, leave address of line) 9 : L'INE DUP OFFFO AND 10 IF . " NOT ON CURRENT EDITING SCREEN" QUIT THEN SCR & (LINE) DROP 3 11 12 13 14 --> 15

```
SCR # 201
  8 ( WHERE, BLOCATE
                                                                       810707 SHO >
  1
  2 UDCABULARY EDITOR IMMEDIATE HEX
  3
4 : UHERE ( print screen # and image of error)
5 DUP B/SCR / DUP SCR ! ." SCR # " DECINHL .
6 SUMP C/L /MOD C/L + ROT BLOCK + CR C/L TYPE
7 CR HERE C# ~ SPRCES 5E EMIT
8 ICOMPILEJ EDITOR GUI' :
  67
  9
 10 EDITOR DEFINITIONS
 11
      RB & C/L /MGD 3
 12 : #LOCATE
 13
 14
 15 ->>
SCR # 202
   & CHEAD, HEAG, -MOUE, BUF-NOVE
                                                                        BLOYOF SHE >
          LEAD ( --- line address-2, offset to cursor-i)
#LOCATE LINE SWAF ;
   2 I BLEAD
   3
   4
        5 #LAG
   7
         MDUE ( move from adr-2, to line-1 ---- )
LINE C/L CMOUE UPDATE :
   8 : -MOVE
   9
 10
  11 : BUF-MOUE
                              ( move text to buffer-1, if any ---- )

    Commentation
    Commentation

    PRD
    1+

    LF
    PRD

    SWAP
    C/L

    1+
    CMOUE

    ELSE
    DROP

    THEN
    J

  12
  13
 14
 15
SCR # 203
                                                                        810767 SHD 7
   0 ( XLINE#, FIND-BUF, INSERT-BUF
   2
3 : XLINE#
                                < convert current cursor position to line#>
          WLOCATE SUPP DROP 3
   Ξ.
        FIND-BUF ( buffer used for all searches )
PRD 50 + 3
   6
7 : FIND-BUF
   8
   á
  10
  11 : INSERT-BUF ( buffer used for all insertions )
12 FIND-BUF 50 + J
  13
  14 --->
  15
SCR # 204
8 < (HOLD-, (KILL-, (SPREAD-, X
                                                                         818787 SHU )
   1
   2 & (HOLD)
         HOLD) ( move line+1 from block to insert buffer )
LINE INSERT-BUF 1+ C/L DUP INSERT-BUF C! CHOVE ;
   5 : (KILL)
6 LINE
7
          ( ename line-1 with blanks )
LINE C/L BLANKS UPDATE :
   8 : (SPREAD)
                              ( spread, making line# blank )
      LINEN DUP 1 - GE
DO I LINE I 1+ -MOUE -1 +LOOP (KILL) 3
   •
 10
 11
 12 I X ( delete line@ from block, put in insert buffer)
13 >LINE@ DUP (HOLD) @F DUP ROT
14 D0 I 1+ LINE I -MOUE LOOP (KILL) ;
 15 --->
SCR # 205
8 ( DISPLAY-CURSOR, T, L
                                                                        810715 SHU )
  1
  2
3 : DISPLAY-CURSOR ( --- )
4 CR SPACE WLEAD TYPE SE EMIT
5 WLAG TYPE WLOCATE . DROP 3
   7
   S : T (twee lined-1)

S C/L + R0 ! 0 DISPLAY-CURSOR J
  18
  11
  12 f L (list current screen )
13 SCR @ LIST #
  14
  15 -->
```

```
SCR 0 296
0 ( N, B, (TOP-, SEEK-ERROR
                                                                               618787 SHD >
   1
   2 I N
                                  < select next sequential screen >
          1 SCR +! J
   ŝ
          4
   518
   67
                                 ( reset cursor to top of block )
   B & (TOP)
      - JEER-ERROR ( Output error mad if no match )
(TOP) FIND-BUF HERE C/L 1+ CMOUE
HERE COUNT TYPE
." NONE" QUIT :
--->
          0 R0 1 3
 10
  11 : SEEK-ERROR
  12
  13
 15 --> <sup>(1)</sup>
SCR # 287
8 ( (R-, P
                                                                                818787 SHD >
   3 I (R)
                                 ( replace current line with insert buffer )
          X.INE#
   4
           INSERT-BUF 1+ SWAP -MOUE J
   5
   6
   7
  7
8 t P
9 SE TEXT
10 INSERT-BUF BUF-HOUE
11 (R) ;
                                 ( following text in insert buffer and line)
  13
  14 --->
  15
SCR # 296
   8 ( WIPE, COPY, ILINE
                                                                              819715 SHD >
   3 | UIPE ( clear the current screen )
4 10 0 DO I (KILL) LOOP ;
   5
           OPY (corruscreen-2 onto screen-1)
B/SCR + OFFSET (E + SWAP B/SCR + B/SCR
OUER + SWAP
DO DUP FORTH 1 BLOCK 2 - ! 1+ UPDATE LOOP
    6 # COPY
   8
   9
  10
          DROP FLUSH ;
  11
  12 : ILINE ( scan current line for match with FIND buffer )

13 ( undate cursor, return boolean )

14 @LAG FIND-BUF COUNT MATCH R8 +! ;

15 --->
           810715 SHD
SEEK) ( FIND buffer match over full screen, else error)
BEGIN 3FF RM & (
IF SEEK-ERROR THEN
ILINE
UNTI -
 SCR # 209
    & ( (SEEK-, (DELETE-
                                                                                 810715 SHD >
    2 I (SEEK)
    3
    4
    5
            UNTIL
    7
         (DELETE) ( backwards at cursor by count-1 )

>R #LAG + R - ( save blank fill location )

#LAG R MINUS R# +! ( back up cursor )

#LEAD + SWAP CMOUE

R> BLANKS UPDATE J ( fill from end of text )
    8 : (DELETE)
    ž
   18
   11
   12
   13
         --->
  14
15
```

.

```
SCR # 210
0 < (F-, F, (E-, E
                                                                            810715 SHD )
   1
   2 1 (F)
                                ( find occurance of following text )
           SE TEXT
   3
           FIND-BUF BUF-HOUE
   4
   5
  6
7 1 F
         F (find and display following text)
(F) DISPLAV-CURSOR :
   8
   q
         (E) ( enase backwards from cursor )
FIND-BUF CO (DELETE) /
  10 + (E)
 12
 IS TE (enase and display line)
14 (E) DISPLAY-CURSOR 1
15 --->
SCR # 211
8 < D, TILL
                                                                            810715 SHD )
   1
   2
  3 t D
4 (F) E J
                              C find, delete, and display following text)
   5
  < delete from cursor to text end >
         TILL ( delete from cu

#LEAD + 5E TEXT

FIND-BUF BUF-MOUE

ILINE 8= IF SEEK-ERROR THEN

#LEAD + SWAP - (DELETE)

DISPLAY-CURSOR 3
   8
   9
  10
  11
  12
  13
  14 -->
15
 SCR 0 212
                                                                            818787 SHD )
   8 < COUNTER, BUMP
    1
   3 8 UARIABLE COUNTER
    4
   5
           UMP ( the line number and handle Pamins)
1 COUNTER +! COUNTER @
38 > IF @ COUNTER !
CR CR @F MESSAGE @C EMIT THEN 3
   6 I BUMP
   7
   8
   9
  10
  11 --->
  12
13
  14
 SCR # 213
0 < 5
                                                                             818715 SHD >
           SIG715 :

( from current to screen-1 for strind )

GC EMIT SE TEXT & COUNTER !

FIND-BUF BUF-HOUE

SCR & DUP XR DO I SCR !

(TOP)

DECT
    1
   2 : S
3
    4
    5
    67
            BEGIN
              ILINE IF DISPLAY-CURSOR SCR ? BUMP THEN
3FF RN 4 <
    8
    9
           UNTIL
   18
           LOOP R> SCR ! J
   11
                                                    .
   12
  13 ->
14
15
```

```
SCR # 214
  BCIJU
                                                                       810715 SHD )
                             ( insert text within line )
  1 2 I
         SE TEXT (load insert buffer with text)
INSERT-BUF BUF-HOUE (if anw )
INSERT-BUF COUNT BLAG ROT OUER MIN XR
R RB +! (bump cursor)
  2
  ž
         R R# +!
R - XR
  5
                                                ( characters to save )
  6
         R - MR
DUP HERE R CHOUE
HERE WLEND + R> CHOUE
R> CHOUE UPDATE
                                                ( from old cursor to HERE )
  7
                                                ( HERE to cursor location )
  8
                                               ( PHD to old cursor )
  9
         DISPLAY-CURSOR /
                                               ( look at new line )
 10
 11
                               C insert following text under current line?
 12 I U
         C/L R# +! (SPREAD) P J
 13
 14
 15 -->
SCR # 215
                                                                         810715 SHD >
   0 ( R, M
   1
   2 I R
                               ( replace found text with insert buffer )
          (E) I J
   3
                               ( move from current line on current screen )
( to screen-2, UNDER line-1 )
   5 8 M
          SCR & XR
R# & XR
XLINE# (HOL
   6
                                < save original screen and cursor location >
                     (HOLD)
                               ( move current line to insert buffer )
   8
          SUMP SCR !
1+ C/L + R#
(SPREAD) (R)
R> C/L + R#
R> SCR ! ]
   9
                              ( set new screen # )
! ( text is stored UNDER requested line )
  18
                               ( store insert buffer in new screen )
! ( set original cursor to next line )
  11
  12
  13
                               ( restore original screen )
  14
  15 FORTH DEFINITIONS DECIMAL
 SCR # 216
   8 <
                                                                         818715 SHD >
   1 FORTH DEFINITIONS DECIMAL
   2 : 20R0P
                                         ( drop a double number )
                 DROP DROP 3
   3
   4 : 25WAP
                                          < 2nd double number to TOS >
   5
          ROT XR ROT R> 3
   6
7 1 2DUP OUER OUER ;
                                          < dum a double number >
   £
   9 : (MATCH)
                                          ( addr-3, addr-2, count-1 -- flas )
           -DUP IF OVER + SUMP
  10
                        DUP COLI COLI -
DUP COLI COLI -
IF OFFILERUE ELSE 1+ THEN
LOOP
  11
  12
  13
  14
                  ELSE DROP OF THEN ;
  15
                                                                                    ز___
 SCR # 217
                                                                          818715 SHD 0
   0 (
   1
           ATCH (curson adr-4, butes left-3, string adr-2)
(string count-1 -- flag=2, curson offset-1)
>R >R 2DUP R> R> 2SWAP OUER + SWAP
(caddr-6, bleft-5, faddr-4, flen-3, caddr+bleft-2, caddr-1)
   2 . MATCH
   4
   5
           00
    6
   7
                20UP I SURP (MRTCH)
                IF

XR 2DROP R> - I SURP - 0 SUMP 0 0 LEMUE

( caddr, bleft, $addr, $len OR 0, offset, 0, 0
   8
   9
                                                                  0. offset. 0. 0 )
  10
   11
           LOOP
2DROP
   12
           2DROP ( caddr-2, bleft-1 OR 0-2, offset-1 )
SWAP 0= SWAP 3
   13
   14
   15
```

RECURSION AND THE ACKERMANN FUNCTION

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 :=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)

© VARIABLE CALLCNT

: ACKERMANN (IJ-F)

1 CALLCNT +!

O= IF

SWAP DROP 1+

ELSE

DUP

O= IF

DROP 1- 1 MYSELF

ROT ROT DROP 1- SWAP MYSELF

THEN

THEN ;
```

(NIC-forth)

VARIABLE CALLCNT

```
: ACKERMANN (IJ-F)

1 CALLONT +! 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 \approx 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, J=6 function

R

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.

<u>K</u>	<u>J</u>	F 1	CALLCNT	MAXDEPTH
Ō	0	ī	1	
0	1	2	1	
1	0	2	2	
1	1	3	4	3
1	2	4	6	
2	0	3	5	
2	1	5	14	
2	2	7	27	8
2	3	9	44	10
2 2 2 2 2 3 3 3 3 3 3 3 3 3 3	128	259	33669	
3	0	5	15	
3	1	13	106	
3	2	29	541	
3	3	61	2432	63
3	4	125	10307	127
3	5	253	42438	255
3	6	509	172233	511
4	0	13	107	16
4	1	??	??	??

SIC-forth is the implementation of FORTH on The NICOLET INSTRUMENT CORPORATION 130/1280 series computers. This computer is a 130-bit minicomputer with a 19-bit address stace.)

> Joel V. Peterson Nicolet Instrument Corp. 5225 Verona Road Madison, Wisc. 53711 (608) 271-3333

E. - A great article, but watch out. Most "FORTH implementations have insufficient mack space to execute this function. Programs rould be reviewed for compatibility.)

REVIEW

A Brief Review of the Manuals for the PET/CBM fullFORTH+ V1.3/4

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 $5\frac{1}{4}$ " 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 "a 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 $\pm LOOP$: "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 increment--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 (not PET's), 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.

FORTH, INC. NEWS

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 wholly subsidiary а owned of Technology, and the present shareholders of become FORTH will 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. "I'm extremely excited about these plans."

EXPANSION 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 <u>Starting</u> <u>FORTH</u>, acting as manager. The publications department has grown by two, and three general support staff members have come on board.

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 raster format.

The raster-scanned maps will be displayed on a high resolution (1024×1024) 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.

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 Generally speaking, however, programs. 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:

> MARKETING COLUMN Editor, FORTH DIMENSIONS PO Box 1105 San Carlos, CA 94070

3

1

i

ŧ

1

1

i

j,

p

۵

a 7

۶

21

HELP WANTED

FORTH PROGRAMMERS

Openings at All Levels At FORTH, Inc.

Programmers 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

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-compiled application. Assembler level familiarity preferred with the 8080, PDP/LSI-11, 8086, M6800, CDP1802, NOVA, IBM Series I, TI990. Communication skills are essential.

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.

SENIOR 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.

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.

JR. INSTRUCTOR

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

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 2K of ROM and 128 nibbles of RAM.

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:

THE COLEMAN COMPANY, INC. Scott Farley Design Project Manager (316) 832-6545

NEW PRODUCTS

FORTH by Timin Engineering, Release 3

Release 3 of FORTH by Timin Engineering is a complete software development system. It is -teractive (conversational) in nature. The -ORTH system incorporates a command processor, compiler, editor and assembler, all memory resident. The principal benefits are a reduction software development time and a reduction memory size for large applications. The crincipal application area has been machine and process control. The language is suitable for all scolications except scientific mathematics. his product is based on the well-known FIG FORTH but with numerous enhancements, -pludina:

- 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

Release 3 of Timin FORTH will run on Z-3C/8080/8085 hardware systems with CP/M or DOS. Minimum memory size is 28K. The store for Release 3 of Timin FORTH is \$235 (if store than 8" standard disk, add \$15). To order Felease 3 of Timin FORTH, write Timin Engineering Company, 9575 Genesee Avenue, Suite E-2, San Diego, CA 92121, or call (714) 455-9008.

HDOS 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.

r -

t

Ł

₽

Ð

5

- Order turnaround time: 3-4 weeks.

Order from:

Essex Computer Science Richard E. Smith 1827 St. Anthony Avenue St. Paul, MN 55104 (612) 645-3345.

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 8K from address 0 is required

Version 2--RCA unit-track Load under RCA unit-track Disc with source and object files for RCA CDP185008 CDP185007 CDP185005 with UART card A minimum of 8K from address 0 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

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.

I will supply my AIM-FORTH "Hacker's System" to anyone for \$25.00. THIS IS NOT FOR BEGINNERS! THIS IS NOT A COM-MERCIAL 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.

Eric Johansson

55 A Richardson St. Billerica, MA 01821 (617) 667-0137 (home) (617) 899-2719 x 224 (work)

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

FORTH CLASSES

NEW CLASS BY KIM HARRIS & HENRY LAXEN

FORTH, PRINCIPLES AND PRACTICES

This class is intended to teach the student row to write programs in FORTH. It is a "how to" class and not a "why" workshop. The class will meet on each Monday in October from 6:30 to 9:30 at Berkeley Computer, 1569 Solano Avenue, Berkeley. The phone number there is 526-5600. The topics to be covered are:

The Language Input Output Structure String Handling Vocabularies Defining Words

This is an ambitious schedule, and depending on the level of the students, more or less will be covered. Experience with other computer languages would be helpful, though it is not required. There will be homework exercises, and machines will be available for students' se. For more information, contact Henry _axen at (415) 525-8582.

SEMINARS, WORKSHOPS, CLASSES FROM FORTH, INC.

_ocation	Seminer	Warkshop
Los Angeles	October 15	October 16
San Diego	October 22	October 23

Introductory classes in polyFORTH crogramming will be offered September 14-18 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.

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.

Workshop Dates	Time	Cost
Sept. 21-25	9-4:30	\$295
Oct. 19-23		
Nov. 16-20		

To obtain more information on these workshops, call Inner Access (415) 591-8295 in Belmont (home of Marine World) in the San Francisco Bay Area.

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.



How to form a FIG Chapter:

- 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 x 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 am.

Southern California

Los Angeles

- 4th Sat FIG Meeting, 11:00 a.m., Alistate Savings, 8800 So. Sepulveda, L.A. Philip 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. Guy Kelly, (714) 268-3100, x 4784 for site.

Northwest

Seattle Chuck Pliske or Dwight Vandenburg, (206) 542-8370.

New England

Boston

1st 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.m., Cochituate, MA. 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, OK. 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) 387-6976. John Hastings (512) 835-1918.

Mid Atlantic

Potomac Joel Shprentz, (703) 437-9218.

- New Jersey George Lyons (201) 451-2905.
- New York Tom Jung, (212) 746-4062.

Midwest

Detroit Dean Vieau, (313) 493-5105.

Foreign

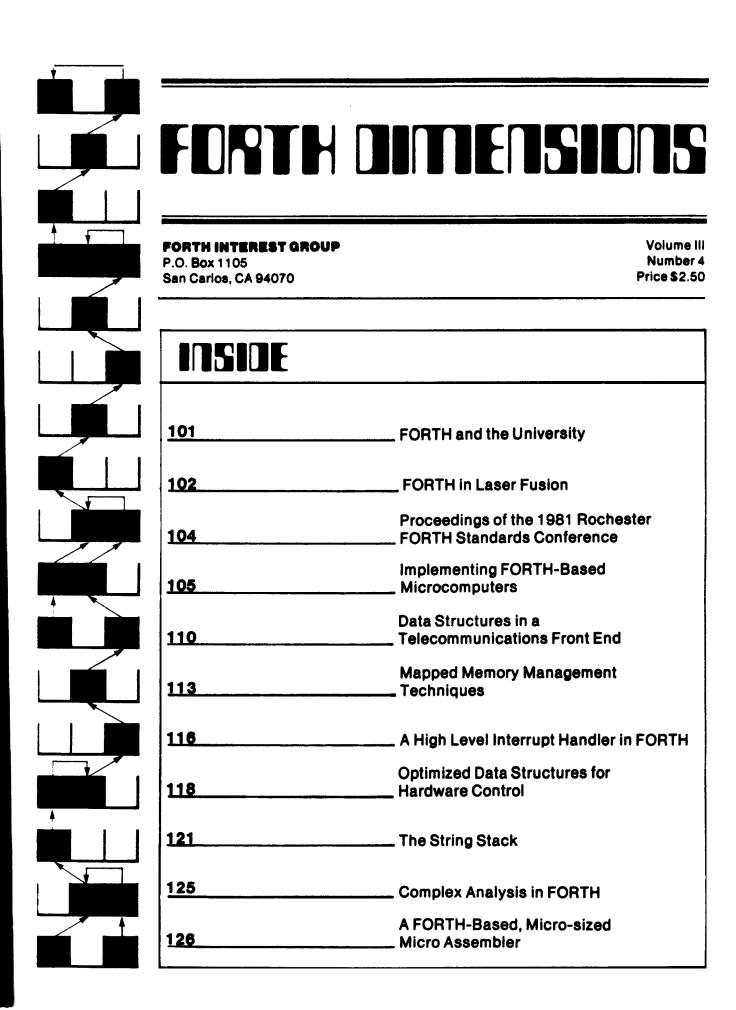
- Australia Lance Collins (03) 292600.
- England FORTH Interest Group, c/o 38, Worsley Road, Frimley, Camberley, Surrey, GU16 5AU, England
- Japan FORTH Interest Group, Baba-bldg. 8F, 3-23-8, Nishi-Shimbashi, Minatoku, Toyko, 105 Japan.

Canada

Guebec 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 DIMENSIONS

Published by Forth Interest Group

Volume III No. 4

November/December 1981

Roy C. Martens

C. J. Street

Publisher Editor

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 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 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

HISTORICAL PERSPECTIVE

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

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

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

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

EDITOR'S COLUMN

A special thanks this month goes to Mr. Larry Forsley and the University of Rochester. The majority of this issue comes from his efforts and those of his asociates. While acting as guest editor for this issue 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 Rochester 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 a 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 DIMENSIONS 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.

1 hope to meet many of you at the FIG National Convention in Santa Clara, California on November 28th. Meanwhile, GO-FORTH and get additional members.

C. J. Street Editor

PUBLISHER'S COLUMN

We are heading into some busy times for FIG. By the time you get this copy of FORTH DIMENSIONS we'll have completed the Mini-Micro Show in Southern California and be deep into the details 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 issue 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 ---------and replaces "Using FORTH" as the book to have about the FORTH language.

Now, a little lecture. We have conducted an 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 is make a copy of the Order Form --------- and have your associates fill in their name and address. If we each get one more person to join we'll have over 5,000 members. Let's do it.

Roy C. Martens

FJ

FORTH AND THE UNIVERSITY

Lawrence P. Forsley Laboratory for Laser Energetics University of Rochester

Welcome to the wonderful world of URTH, or, University of Rochester FORTH. URTH was developed several ears ago and has been used for many abolications, some of which are bocumented here. Beginning with the URTH Internatinal Standards Conference, held on Catalina, we have 'c..owed the FORTH standardization effort. As a result, the majority of our sistems are close to being FORTH-79 Standard, although not FIG model. Very 'ew papers in this issue will refer to URTH,

The 1981 Rochester FORTH Standards Conference was held at the University. re major reason for this, aside from the ce .ghtful weather at that time of year, is re FORTH activity at the University. This work shows up in several divisions and separtments including the University Computing Center; Optics; Physics and - stronomy; Chemical Engineering; rechanical Engineering; Department of Fad.ology, Division of Diagnostic Ultra-Stund; Department of Cytopathology; Electrical Engineering and the Laboratory - Laser Energetics. Indeed, we are to the original work by Dick Early, who in 1976 was an assistant profes-= of Physics and Astronomy, for deriving Te first URTH system; and to Ken mandwick, who in 1977 was with the -riversity Computing Center, for bringing = the IBM 360/65 TSO version based on Tak's work. At this time, Ken, Dick and I II hed by Ken, although Dick was partial PARTH, for Mike Williams' - titasking Intel 8080 FORTH system. -- fortunately, Ken and Dick are no longer - the University; and Mike's commit-Tents prevented his authoring a paper. -wever, their work is reflected in the -aterial presented here.

t

ł

Ł

J

5

£

R

<u>}_</u>

æ

ic re

p.

魡

1

-

rh.

This issue starts with three overview =cers. The first paper is mine and covers = development of FORTH at the Laborato for Laser Energetics, which remains<math>= largest university FORTH user. The = cond paper, by Peter Helmers, reflects = the uses of FORTH in medical research = c clinical applications. The third, by<math>= con Lefor, covers one of the more visible $= r \cdot ersity FORTH systems: The IBM 3032$ = ecommunications front-end.

The next three papers demonstrate a set of ways by which FORTH can be used to interact with hardware. The first racer, by Rosemary Leary and Carole Takler, deals with three methods of using Tacced 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 Stack. 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 floating point package with words appropriate to the complex plane. The String Stack is clearly a system tool. Complex arithmetic is less so, and a microprogramming system is clearly an application. Or is it? In the context of <u>its user</u>, the microprogramming words are a system. We seem to be forever chasing our tail when determining a FORTH context. But I 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,¹ is used in both Joe Sawicki's and Greg Cholmondeley's code. Mike McCourt's "IN" concept² 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 European FORTH User's Groups and FIG have all played this role. But another view is possible, which is more in keeping with FORTH's 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 among users. I suggest that this direction should be one of form, and not of content.

It is appropriate to define documentation standards which imply a form. But is is inappropriate to state that something can be done 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 found in a conversation I had with Kim Harris.² Kim took exception to an earlier paper by Peter Helmers on Userstacks.⁴ I was told that the approach was wrong. Period. But on further discussion, I found that I 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 multitasking systems, as his systems are all single user, interrupt/event driven. thus, it is worth remembering that each of us has different, and valid, viewpoints.²

As a major promoter of FORTH at the University of Rochester, I have tried to define an 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 with university teaching as those two languages did. In five years my present students will be in industry, as my first student contacts already are. A university 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.

- 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 <u>Proceedings of the 1981</u> <u>Rochester FORTH Standards Conference.</u>

^{1.} FORTH DIMENSIONS Vol. I No. 4 and Vol. I No. 5.

FORTH IN LASER FUSION

Lawrence P. Forsley Laboratory for Laser Energetics University of Rochester

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 techniques, like FORTH, for enhancing interactions between engineers, physicists and their experiments.

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 turn, 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 systems on the 24 beam 13 terrawatt infrared Omega laser and all of the computers on the single beam Glass Development Laser (GDL) 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 students, of LLE and the University. Most of our FORTH systems have been partially, or totally, implemented by students from chemistry, electrical engineering, physics and computer science. Four of the other papers in this journal issue have a student author who is also a member of LLE.

Standardization

LLE was one of the first Laser Fusion laboratories to automate its laser systems. 1 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.

languages, Computer including FORTRAN, are different from one vendor to another, and especially when operating system calls were taken into account. The problem of software consistency 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, a 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 (12:1991-2019) 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.

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 (10:223-240) at the Kitt Peak and NRAO facilities which are funded by the National Science Foundation. I found three groups at these facilities using FORTH: scientists, computer engineers and technicians. In some

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 NRAD's 36 foot radio telescope. He was pleased with his ability to change the graphics routines and other "systems" software while continuing to collect data. Similarly, I found many technicians 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 with 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 maintain programs in this environment.

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 documenting 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 Pesk Varian nucleus circa 1974. He recoded it for the National Semiconductor PACE microprocessor. Ken Hardwick, then with the Univerity Computing Center,² used this as a model for the IBM 360/65 under TSO and Mike Williams developed a 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 demonstrate an automated X-Ray spectrometer. Although I had a system for the Hewlett Packard 2100 from Kitt Pesk 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 University community and the Laboratory for Laser Energetics have worked on various versions of FORTH for Data General, Modcomp, PDP 212 and IBM 3032 compu-Through the efforts of Mike ters. McCourt, originally with the Department of Cytopathology and then with LLE, we developed a FORTH-79 system. All of these were multitasking systems (2:314-

Z

118).

Testbeds

The first FORTH applications at LLE were hardware testbeds. There are two sistinct phases in dealing with hardware. The first occurs during its initial checkout and reoccurs when it fails, or you suspect 1 of failing. At this stage, one is concerned with device and interface implementation, and it is important to be able to interactively set and test data and address lines.

A testbed must be capable of exercising hardware at a rate of about 1 kiloretz. Devices which operate in a faster time domain will usually be buffered, as an example, with transient digitizers. Most other devices, such as relays, operate in a 10 Hz or slower time domain. At a 1 kHz rate, sufficient samcles can be taken from A/D's and D/A's to suickly check their accuracy and range, and thereby checkout many parts of a system quickly.

Several language features are required for tests like these. A means must be proided to individually and collectively set address and data lines. There must also be a way of repetitively issuing data/ address catterns. Often, a hardware problem is intermittent, and a test and branch capachity is necessary to allow loopiung until a failure occurs.

Thus, the specification for a testbed anguage grows quite large, with a major to eccupied by the command processor, at text interpreter. Regardless of whether the testbed language is implemented in Fortran, Basic, Pascal or most ther programming languages, a substantial effort will be spent on the text interpreter. One of the virtues of FORTH is that it comes with a generalized text atterpreter, suitable for testbeds and ther applications.

Our FORTH testbed applications in-...ded: power conditioning testbed for mecking out laser amplifiers; alignment testbed for debugging and calibration of sutomated components; and, general LAMAC module testing. Other testbeds have been used to develop image protesting hardware and software, and onetimensional reticon arrays.

The laser amplifier testbed was reveloped along the following schedule:

- 1. October 1977-Ken Hardwick and I began writing a FORTH system for the HP 2114.
- January 1978- The FORTH system was completed and CAMAC software started.

- 3. March 1978- A laser amplifier testbed was demonstrated.
- April 1978- Single laser amplifier testbed 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-beam milestone was originally scheduled for early September, 1978, this left insufficient time for laser preparation.

- 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 system was operational.
- 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 16K words of memory and a floppy disk, the Fortran based system required 196K 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, I 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.

Spatial and Temporal Relationships

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 subsystems. 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 discussions with Ray Helmke and Eric Knobil at the Wilson Synchrotron,⁴ lad me to develop such a language for process control called Maps, because it "maps" relationships 6:109.110.

A Map contained two types of structures, 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 21MX-E computer. This computer currently runs 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 database-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.

Production Systems

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 alignment procedures more quickly. However, there was a risk asso-

5

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.⁵ 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.⁹ Unlike those systems though, this was originally not a turkey system provided by software engineers, but rather was incrementally developed by physicists and students.

We also use FORTH exclusively on the Glass Development Laser (GDL) 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 11 is used in a PAR OMA for processing streak camera data. This is especially significant since GDL is engaged in converting the infrared light to ultraviolet, and the first harmonic IR, a second harmonic green and the third harmonic, UV are observed with the same streak camera. This required a 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.⁷ Finally, an LSI 11/23 is used 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.

Conclusion

Although FORTH was relatively unknown, it has made a positive impact on the development of systems and instrumentation at LLE. It has allowed the computer sytems group to adopt the philosophy of providing tools to scientists and engineers, equipping them to do a 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 GDL, and a fair number on the Omega systems and other laboratories at LLE, FORTH has been a success.

Acknowledgements

I would like to thank an almost endless list of people for their help over the past five years. Most important 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 sytems 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-AC08-80DP40124.

Lawrence P. Forsley is group leader of the Computer Systems Group at the Laboratory for Laser Energetics, University of Rochester, Rochester, N.Y.

Footnotes

- I The four-beam system, Delta, had computer control and monitoring in 1972. (6:101).
- He is now with the Defense Mapping Agency in Washington, D.C.
- 3 Kan is now with Network Systems Inc., in Minnespolis, MN.
- ¹ Cornell Univerity in the summer of 1977. This facility is now known as the Cornell Electron Storage Ring.
- ² The mapped memory techniques 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 Handler in FORTH", which can be found in this issue.

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 Catalina 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 everyone of the seventy participants presented something, and no one missed anything (we think).

In addition, we added travel sponsorship 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 adopted combined poster sessions, oral sessions and some material not presented at the conference. There is an entire section devoted to working groups on areas like Standards clarification, FORTH techniques, Floating Point and Files Management. There are 378 pages covering the state of FORTH. The Proceedings are available 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 area will be <u>Process</u> <u>Control and Data Acquisition</u>. 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. Forsley Laboratory for Laser Energetics 250 East River Road Rochester, NY 14623

٢

c

:

5

Э

е Ч

,

Ē

.

2

•1

1

14 79 78

IMPLEMENTING FORTH BASED MICROCOMPUTERS AT THE UNIVERSITY OF ROCHESTER MEDICAL CENTER

Peter H. Helmers

Introduction

"The micros are coming." Everyone has heard this so that it is not unexpected that physicians and researchers at the University of Rochester Medical Center ask the question: "How can they be put to use?" Over the past four years I've been attempting to answer this question by assembling a series of microcomputers for both research and clinical applications. These systems are all similar in their use of an S-100 bus hardware architecture and a FORTH software environment. Yet they infer significantly when it comes to becific hardware interfaces, application software, and types of system users.

In this article, I am going to focus on both these similarities, and these differences in microcomputer systems. I am coing to start out by discussing their common hardware foundation, and then explore peripheral devices unique to each s-stem's design. Because the ultimate sers of a system have a significant mpact on application software, I am going to try to characterize the types of users I "ave dealt with, and their specific soft-+are capabilities and needs. From here I +.11 discuss some common software packages that were written to transcend both variable hardware, and variable user, requirements. By discussing all of this in terms of how FORTH has aided system sevelopment, I hope to fully support my contention that FORTH is an ideal enviconment to meld many different types of _sers to just as diverse hardware configu-*ations.

General Hardware Organization

So let's start out by considering the common architectural arrangement of inese microcomputers. They are all Z-80 cased machines with typical memory sizes if from 32K to 48K bytes of static read/ enter memory and 1K to 2K of EPROM memory used to contain machine specific mplementations of commonly needed I/O mouthers such as console and disk drivers. Each microcomputer uses one or two eight inch single density floppy disk drives. The cumary system console is comprised of a is line by 64 character memory mapped is do display along with detached ASCII exboard. Each machine also has an RS-132 serial port for printer hookup.

These computers are all organized around the S-100 (IEEE-696) bus with from ten to fifteen card slots available. With the basic setup described above using from four to six of these slots, the customization to specific system configurations is accomplished by a mixture of standard commercial and/or wire-wrapped peripheral interface cards. Let's consider some of these systems in greater detail, looking at special hardware and how this is reflected in the systems' software.

Ultrasound Diffraction Apparatus (UDA)

The UDA microcomputer is part of an experimental system to explore the scattering (diffraction) of medical ultrasound The signals through tissue samples. scattering is a function of both frequency of the ultrasound signal (2 to 8 Mhz) and the angular position of a receive transducer 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 a Hewlett-Packard 8165A programmable signal 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%.

Study of Vein Mechanics

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 pressure 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 up 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 thresholding, then detecting vessel edges via a software algorithm. By using FORTH/Z-80 assembly language, the diameter determination executes in less than one second.

This data acquisition system also contains a dual mode graphics display capable of 128x128x4 grey scale images or 256x 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 a dot graphic plot of TV signal amplitude versus time.

Also included in this system, to aid in data reduction, is an Advanced Micro Devices AM9511 high speed floating point processor IC. This circuit's speed, combined with the memory mapped graphics display, allows <u>real-time analysis and display</u> 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 data collection and analysis. Now data can be automatically acquired and processed within a couple of hours.

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 capillary 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 (C_2H_2), 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.

X-Ray Scanning System

This experimental scanner uses a slotted wheel and two horizontal slots (mounted at 90° 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 attenuation and X-ray power), and an eight bit D/A converter to control the exposure time of each X-ray pulse. Several digital 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.

How Users' Needs Impact These Systems

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 benefits they may have used FORTRAN on a large machine for number crunching, but otherwise they have few preconceived notions about computer organization. They are less impressed with structured programming techniques or file systems than they are by the fact that they can physically, and interactively, control peripheral devices. A research scientist may notunderstand how a word like RAMP or SAMPLE works, but can readily learn what they do.

For example, the FORTH software written for the UDA system 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:

OK 25 DB

(RFN's a natural herei) OK * FRQ 2500 KHZ* TALK

(via the GP-IB)

OK 2.5 USEC CARRIER-OFF

A data acquisition experiment can be set up using words such as:

OK 100 2000 SWEPT-FREQUENCY (define control of HP8165A) OK FIXED-ATTENUATION (define control of atten) OK DON'T-SHOW-ATTENUATIONS OK 1500 32 NOVA-CONTROL (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 Xray scanner system can be easily programmed by:

OK MOTOR WHEEL-MOTOR (define a 'MOTOR' data type) OK : ROTATE-EM OK 00 WHEEL-MOTOR RAMP OK (ramp stepping motors) LIMIT-SWITCHES? OK (exit loop if motor limited) **OK** SYNCHRONIZE (synchronize to motor pulse) OK LOOP

<u>ok</u> ;

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 specific ways towards some specified clinical objectives. Thus they need to implicitly use both basic FORTH words and peripheral driver words, but want to only explicitly know words that achieve specific 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:

OK FULMONARY CALCULATIONS (acquire data, and calc it) OK PRINTER SHOW RESULTS (print results) <u>OK</u> DNE VIEW (view plots of gases on) <u>OK</u> C2H2 VIEW

(... graphics display)

By learning a limited, yet full, vocabulary of perhaps 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 apecific usage patterns.

Common Software Packages

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 support, 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 packages with this in mind I have been able to avoid constantly recreating software because of hardware variations.

Common software packages can do more than just ease support for similar systema. It can effectively hide hardware details from the user, thus making dissimilar A/D converters, for example. appear identical from the software point of view. And a well designed set of driver software also imparts increased capabilities to a system 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 these A/D's have eight 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 a 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 1

3

3

1

•

5

.

J

æ

3

3

.

.....

Ń

2

Ē

.

N C

-

H

西南 医发育

1

butput format are hidden from the user. Similarly, the real time clock works in a manner transparent to hardware specifics. DELAY requires only an input argument to specify the number of real time clock "ticks" to delay.

But the conceptual basis of the data acquisition package transcends just the 2^{10} hardware; there must be some place to put the data. This may be on the parameter stack, in data arrays, or in disk based virtual arrays. When this capability is added, the data acquisition specific mardware creates a synergy with the fundamental system hardware such as read/write memory or floppy disk.

Another example of a peripheral driver package that I developed is a memorymapped video graphics package. The 1.pical hardware interfaces ranged from 140x256 resolution up to 512x480 resolution, with as many different methods of addressing specific dots on the display.

Conceptually, we want, first of all, to be able to plot physical X,Y points indecendent of hardware specifics. A word such as PLOT, using X and Y integer parameters on the stack top, can give us this capability very readily.

But to really use graphics effectively, t is nice to be able to specify different areas on the video screen to plot different cata, as well as scaling functions to adopt ogical coordinates to this specified paphics area. The GRAPH data type built with a defining word) allows these afferent graphics areas and scaling functions to be associated, and invoked, by a common name. Further capabilities were added to allow easy creation of vectors, tick marks, axes, and boxes. All of sudden, a very proletarian graphics pericheral is transformed into a powerful tool. And because these new functions are a. built on the PLOT word, they are readily tansferred between systems with : fferent hardware interfaces.

A final software driver to consider is that of the hardware floating point unit. it is interesting to consider this from both . FORTH, and a conventional language coint of view. In a language such as PASCAL, the system generally has built in software based operators for floating count. Because the system is not inherent-· extensible, the addition of a hardware floating point peripheral requires either a anufacturer rewrite of the PASCAL foating point routines, or else a user sterface through PASCAL functions or procedures. The former requires manufacturer acceptance and support of a new -ardware peripheral; unless a very popular zevice, such support will be reluctant at cest. The latter requires a very awkward anguage syntax to invoke hardware floatng 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, a hardware floating point package was easily added as an extention to the basic FORTH system; the language adopted the hardware!

Anachroniam or Portent?

At this juncture it is valid to ask if FORTH justified itself in its use at the University of Rochester Medical Center. Is it an anachronism of the past, or a philosophy portending the future?

Admittedly, FORTH is somewhat limited without such things as a file system or procedural name scoping of variables. Perhaps there should also be less explicit knowledge of addresses, and more system security. 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 repid 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 philosophy is that programming is <u>not</u> 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 a series of words to calculate and graphically display the results of a mathematical analysis. While the apries of capabilities needed will always vary between different systems, it is only by providing a rich enough vocabulary that a user can 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.

Helmers' article continued on next two pages

BUG FIXES

Correction to FEDIT

Sorry you had trouble with FEDIT. The listing was retyped at FIG and several 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 /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 50: + ALIN should be +ALIN

You are perfectly right that source text should be loadable. I talked to some of the people at FIG about this and they were acutely aware of the problem but they are simply not set up to directly reproduce listings 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 fig-FORTH line editor definition, R. I used the pseudonym because FEDIT was the first program I wrote in FORTH and I wasn't really 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. Let me know. Good luck.

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

e L

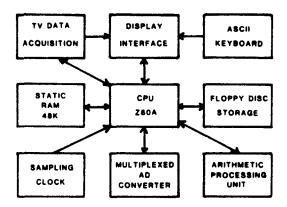


Fig. 1: Block diagram of a typical S-100 based microcomputer; this one is used to study blood vain mechanics.

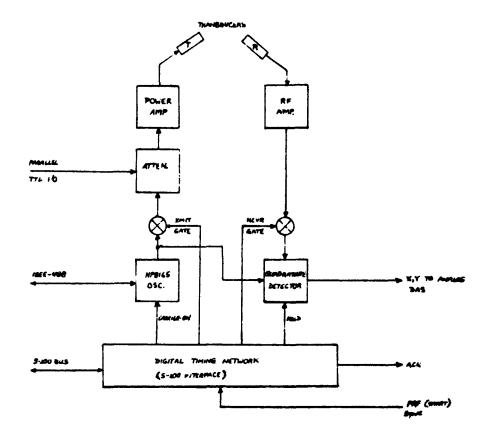


Fig. 2: Block diagram of UDA analog electronics timing control interface. Microcomputer sets up interface parameters, but timing then runs independently using PRFSYNC and ACK handshaking signals from Nova Minicomputer data acquisition system. Because the microcomputer can synchronize to timing hardware, other capabilities such as attenuetor and frequency control can be utilized.

Ŧ

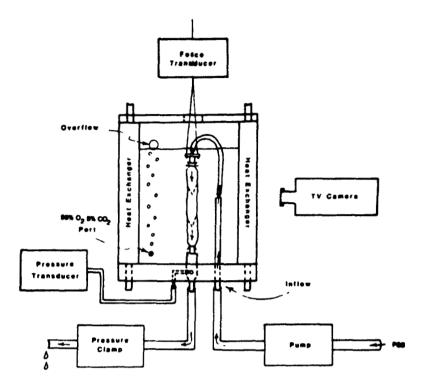


Fig. 3: Diagram of vein mechanics experimental chember. Microcomputer samples pressure and force signals, and determines vein diameter from software analysis of TV image.

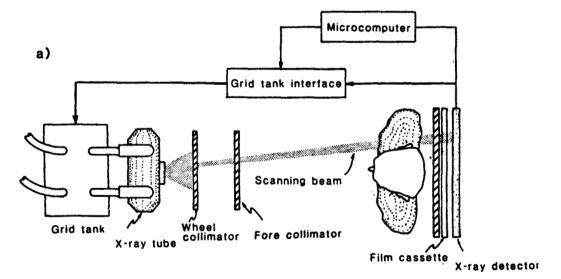


Fig. 4: Diagram of X-ray scanner apparatus showing how wheel collimator and fore and aft horizontal collimators, controlled by stepper motors, create a mechanically scanned X-ray raster. The microcomputer, with A/D and D/A interfaces, also monitors and controls X-ray exposures.

DATA STRUCTURES IN A TELECOMMUNICATIONS FRONT END

John A. Lefor University of Rochester

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 9.6Kb. 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.

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 designed 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 possibly meet these goals and be reasonably cost effective had to be one of local designmeeting 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 (non-SNA) 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 so 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 running 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 appears to be a miracle but represents some faith that the concept is at least essentially sound.

Hardware Decisions

In order to implement the telecommunications front end to an IBM computer, the processor chosen for the implementation had to provide the capability to interface to an IBM byte multiplexor 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 affecting the front end. To this end, the mini-computer chosen as the front end had to have both a history of reliable service and a maintenance team 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 (actually many of the terminals are supported by a Digital Communications Associates 205, which emulates 32 lines of DZ-11 on a single 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 most notable is in the maintainability of the DX-11B (the channel 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 operation. 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.

3

11

e

٦

3

3

8

3

3

3

: A

3

=

3

-

T

Software Decisions

In determining the nature of the software to run for this application, it was necessary 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 I/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 .mplementation which provides support for reltime device handling.

The wide variety of I/O devices which were contemplated for the front end also regired that the software provide tools to help 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 channel interface which had no well defined characteristics (the best documentation of how the DX-11B worked was found in the diagnostic programs supclied for hardware maintenance). In addition, there was always the possibility of needing to support a new and different class of I/O device. Though the manuals socumented how the hardware worked, any software which would allow interaction with the unfamiliar hardware would be beneficial in the debugging of the overall system.

Another area of debugging which was considered in the software choice was the software protocols. The connection to the channel of an IBM computer by asynchronous ASCII devices invokes a nontrivial set of software protocols. A simple example of the kinds of problems is in the transmission of any single ASCII character to the channel. In the IBM environment, the software running in the processor expects that any ASCII characters transmitted from a telecommunications front end are sent not as simple ASCII characters (as generated by the terminal), sut rather demands that each ASCII char-sater be bit reversed. Though this is not a difficult feat to accomplish, it points but the nature of some of the software protocol issues which must be dealt with a telecommunications front end. Suffice it to say the software used to cesign the front end would benefit the sesigner if it helped to identify, and esolve, software protocol issues.

In the development of any realtime software project, it is recognized that the inroughput of the system is important. The telecommunications front end is no exception. Since there are to be a large rumber of I/O devices providing input to the software application asynchronous to the operation of the software, it is imperative that the application software be able to keep pace with the demand. On the ther hand, the inability of the front end to keep pace with the demand is not critiral. If a character destined to a terminal s lost, a human being will not die but a programmer may get upset. Keeping nese priorities in mind the project had to re implemented in an environment which -as not wasteful of processor time, but there was no need to be alarmed if there was the potential to loose data.

The hardware decision made specific features of the processor had to be considered in the software choice. Specifically, the PDP 11/34 had 64K bytes of memory. We had to have some degree of confidence that the entire system could be packaged in 64K bytes. If that was not possible, the development time could be slowed down waiting for shipment of additional memory. The speed 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 end on a processor which was not in use. That was a real opportunity not to be passed up. However, the processor 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.

Alternative Software Strategies

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 outlined. It tends to be compact in memory 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 Thus, development of a programmer. 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 learning curve to train a new individual.

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 sophisticated 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.

Evaluation of URTH

URTH appears to meet many of the goals 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 afforded by URTH is implementation Most of the other advantages protime. vided by URTH can be directly tied to the speed of implementation. URTH provides easy access 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. 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 fault. 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 environment 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/O 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

Ł

5

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.

Alternative Design 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 sufficent 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 so 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 idea to define a "named object stack"⁵ for each I/O entity defined in the telecommunication environment. When a particular I/O device needs some form of service, the named stack is invoked and all data relating to the I/O device is availa-The stack can contain pointers to ble. ring buffers as well as current status of the device. Using this strategy provides 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 many old functions can take advantage of the data structure.

The stack will contain sufficient volumes of information about each I/O 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 a convenient notation.

Summery

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 a complex system. It provided a comprehensive interactive debugging environment with the ability to address specific speed inefficiences in a uniform manner. The major drawbacks to the URTH environment resulted from the choice of data structures for intertask communication within the application.

URTH does provide tools to develop the optimal data structures for any particular application. In the case of realtime applications, the choice of data structures is particularly critical. From my experience, I 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 so prevelant in global data structures such as arrays.

A second feature which would be valuable in an 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 realtime applications, but care in the design of data structures should help to make the overall maintenance of the application a simpler chore.

Footnotes

- 1. This is not simple an example of a perverse IBM, but instead is another fact of IBM computing history. The standerd device IBM used to connect ASCII terminals to the host (a 270x) 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 characters to be captured in bit reverse order. Rather than correcting the problem in the front end, they transmitted the bit reversed ASCII to the host, and translated the bit reversed ASCII to EBCDIC for processing. The software stayed, so the need for bit reversed ASCII exists today.
- 2. This advantage was certainly realized in the actual project. The basic system was 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 so many "words" which were already correct, we began to believe that it is very difficult to debug a working program.
- Converting most of the queues to individually assigned ring buffers speeded up overall processing by 20% or more.
- See Peter Helmers, "Userstack", FORTH DIMENSIONS, Vol. III, No. 1 and Peter Helmers, "Alternative Parameter Stacks", Proceedings of the 1981 Rochester FORTH Standards Conference.

Acknowledgements

I would like to thank Richard Marisa, Ken Hardwick, Mike Armstrong, and Mike Williams for their assistance.

J.A. Lefor was senior systems programmer at the University Computing Center at the University of Rochester and is now Data Communications Manager. 1

2

z

e M

MAPPED MEMORY MANAGEMENT TECHNIQUES IN FORTH

Rosemary C. Leary Carole A. Winkler Laboratory for Laser Energetics University of Rochester

Abstract

Three techniques for using memory management hardware in a FORTH system have been implemented at the Laboratory for Laser Energetics at the University of Rochester. One method uses mapped memory for data storage by creating a data window" in the logical address space. A second method increases the available space for programs by mapping tasks in a multi-tasking system. The third uses mapped memory for data storage by taking advantage of special instructions and a second set of memory management registers.

introduction

The problem of insufficient memory for programs or data is commonly encountered on computers with a 16 bit word s.ze. Many manufacturers now offer hardare to alleviate this problem. At the university of Rochester's Laboratory for laser Energetics we have devised solutons to three different aspects of the croblem using FORTH on PDP-11/23 and PDP-11/34 computers.

Two applications at the Laboratory had a need for large image processing arrays to 100K words). We solved this by .s.ng a double precision array index which maps physical memory into a logical memray "data window" within the FORTH system.

On a different, very large FORTH apsection, we needed both more program mace and more data space. We increased the amount of program space by implementing a multi-tasking system in which tertain portions of memory contain the fulleus and common code, while other tections are task specific and are periodcally switched in and out of active use.

To increase the available data space •e are using special instructions and a second set of memory management registers on the PDP-11/23 and PDP-11/34 computers.

Additional material on these systems tan be found in "FORTH in Laser Fusion," to Larry Forsley, in this issue of FORTH CIMENSIONS.

-lardware

The memory management hardware on the PDP-11/23 and PDP-11/34 computers consists of two sets of registers that map the bit logical addresses into 18 bit physical addresses. 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 processor status word.

Each set of registers contains eight 32bit Active Page Registers (APR's). Each APR is actually two registers: the Page Address Register (PAR) which contains a base address, and the Page Descriptor Register (PDR) which contains the page length and the access control key.

The 16-bit logical address space is divided into eight "pages" shown in Table 1. When the memory management unit is enabled, any access to memory will be mapped through the APR for that address.

Page Logical Address Range (octal) 0 0 - 17776

*	20000	-	3///6
2	40000	-	57776
3	60000		77776
4	100000	-	117776
5	120000	-	137776
6	140000	-	157776
7	160000	-	177776

Table 1. Logical Address Space.

The physical memory address that will actually be accessed is a combination of the logical address 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 address. Bits 5-0 of the physical address are the same as bits 5-0 of the logical address. Figure 2 shows the logical address space.

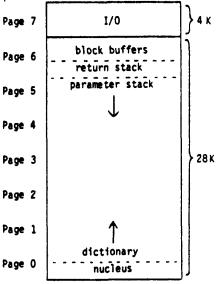


Figure 2. Logical address space for single task without mapped memory.

Additional information on the PDP-11 memory management unit can be found in the processor handbook¹.

Deta Window and Memory Management

One way to utilize the memory management hardware and additional memory is to use it for data storage. Two of our applications at LLE require large data arrays (up to 100K words) for image processing. We solved this problem by creating a "data window" in our logical address space. Figure 3 shows the logical address layout of a system with a data window.

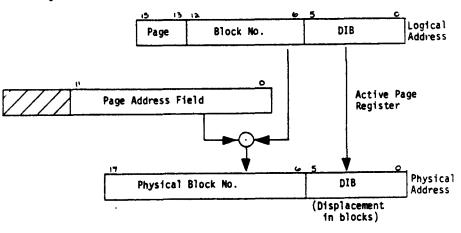
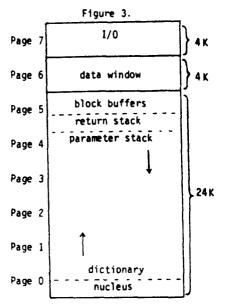
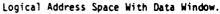


Figure 1. Construction of a Physical Address

(derived from figure 7-9 of [1] and reprinted with permission from DEC.)







The block buffers, return stack, and parameter stack are moved down to the top of the next 4K word page of logical memory, leaving a 4K word gap in the logical address space. In a 128K word system, 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 pixel into the page.

The relocation constant for the needed page is set in the PAR so that it can be accessed through the data window. The logical address of the pixel is obtained by adding the address offset to the starting address of the data window.

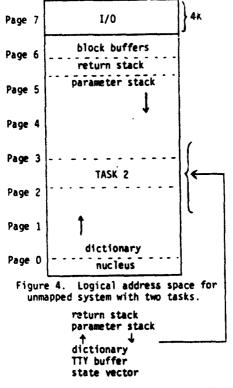
Multi-tasking and Memory Management

Our version of FORTH implements multi-tasking in the following manner. Each task has a "state vector" which contains "user" variables that can differ from task to task. This includes:

- Dictionary and stack pointers
- Program counter and interpreter pointer
- Status flags and state indicators
- Terminal I/O routines and buffer pointers
- Vocabulary pointers
- Number base

The state vector for the master task is included in the nucleus.

Each task also has its 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 shows the logical address space in an unmapped multitasking system.



Task state vectors are linked to each other in a circular fashion, one pointing to the next and the last back to the first. A "round robin" scheduler starts running a new task when the current task executes a PAUSE. PAUSE stores the current machine state into the state vector of the existing task and sets the new machine state according to the new task's state vector.

Additional information on multitasking can be found in works by Forsley², *McCourt*³, and Leary and McClimana⁴. Figure 2 shows the logical address space of a FORTH application with a single task and not using memory management.

To add program space to our multitasking system, we reserved a "task window" in the logical address space. The master task occupies the low five pages of address space. Code in this area is usable by all tasks.

Mapped tasks occupy pages 5 and 6 of the logical address space. Definitions and data within a mapped task are accessible only to itself. Each task must have a separate vocabulary. If definitions in a mapped task are entered into the FORTH vocabulary, the dictionary links will be gone when the next task becomes active. This usually results in a system creat. Figure 5 shows the logical address space in a mapped multi-tasking system.

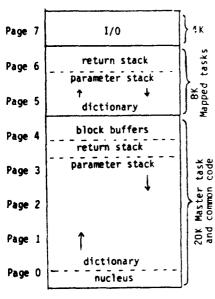


Figure 5. Logical Address Space for Mapped Multi-tasking System.

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 as swapping in the usual state vector information.
- Move the block buffers and master task stacks to the top of page 4.
- Change the routine BLDTASK to assign the new task's return stack, parameter stack, and dictionary to pages 5 and 6, instead of giving them space in the master task's dictionary.
- Change BLDTASK to assign physical 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.

User Space for Data

The two approaches discussed previously both ran in processor "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 processor status word has two mode fields: current mode and previous mode. The instruction MFPD moves a word from the "previous" mode address space to the "current" mode processor stack (the return stack in our FORTH implementation). The instruction MTPD moves a word from the "current" mode processor stack to the "previous" mode address space.

Using these instructions it is possible to retrieve and store data quickly and eff.ciently, and the data stored there is accessible to all kernel mode programs, whether they are mapped tasks or not. Data tables that would otherwise need to be disk resident because of their size can taw be memory resident to speed response time.

The source listing of the user mode tata storage code is included at the end of the sarticle.

Conclusion

The first technique, the data window, tas been used for two image processing accircations. One is used to view infrared and ultraviolet laser beams in materials it mage testing experiments. The system ties circular averaging and calculates an activate intensity within the 10 minute interview.

The other image processing application itserves X-ray diffraction patterns proliced by a nanosecond X-ray source. A econique of radial averaging is also used tere to enhance the diffraction pattern at study changes induced by sample stimston.

The second and third techniques are led on the Omega Alignment System, To how has 17 tasks installed and uses month 140,000 bytes of memory for promemory space. The user mode data storage rethod is used by the data base software att for the intertask message queues.

Although this paper describes techtues used with DEC PDP-11 series comlaters, the techniques are similar to those set with any limited address system with to cal/physical mapping hardware. Thus, the are applicable to minicomputers like the Hewlett-Packard 1000 series and the tach newer 16 bit microcomputers like the Motorola 68000 and Zilog 8000. The storiques are especially appropriate in a IRTH-79 context where the FORTH technic is defined as having a 64K byte access space, carved out of an arbitrarily arge physical address space.

Acknowledgements

4

D

The following people played a major T a in the development of the software Perribed in this article: Donald P. MTClimans, Lawrence P. Forsley, Reade Stimick, Robert D. Frankel, Joseph A. Hate, and Robert L. Keck.

This work was partially supported by the following sponsors: Exxon Research at Engineering Company, General Electhe Company, New York State Energy Essearch and Development Authority, witheast Utilities, The Standard Oil Estimate Utilities, The Standard Oil Estimate (Ohio), the University of Estimates Electric Energy Essearch Corporation, and the U.S. Estimate of Energy inertial fusion estimates under contract number DE-AC08-±2040124.

******************* BLOCK . 445 **************** (MEMORY MANAGEMENT - U@, U!) CODE UP (CADRS]---EDATAJ RETRIEVE FROM USER MODE MEMORY) (SET PROCESSOR STATUS WORD:) 7777760 84 300000 4 MDV+ CURRENT=KERNEL, FREV=USER) FROM ADRS ON STACK TO RP) S @)+ FPD. (PSW BACK TO NORMAL) 7777760 20 HOV. 0 🔹 ((RP TO STACK) (RETURN) s -> RP)+ MOV. NEXT. CODE UI (EDATAJEADRS) RP -) 2 S I) --[] STORE IN USER MODE MEMORY) Mov, (Data From Stack to RP) SET PROCESSOR STATUS WORD:) MOV, 7777760 8\$ 300000 \$ (CURRENT=KERNEL: PREV=USER) FROM RP TO ADRS ON STACK) TPD. S @)+ ((PSW BACK TO NORMAL) MOV, 777760 @\$ 0. (RETURN WITH CLEAN STACK) POP J. ******** BLOCK # 446 *************** (MEMORY MANAGEMENT - K>U) CODE NOU (IN ADRESIDU ADRESIDCOUNTS---ES COPIES 'COUNT') (WORDS FROM KERNEL SPACE TO USER SPACE) (W=COUNT) IJ 5)+ HOU. (RO=USER SPACE ADDRESS) FO S)+ HOV. (RI=KERNEL SPACE ADDRESS) 61 S)+ MOV+ 7777760 81 300000 \$ HOV+ (SET PROCESSOR STATUS WORD:) CURRENT=KERNEL, FREV=USER) (BEGIN, (FROM KERNEL SPACE TO RF) 6F -) R1)+ HOV, (FROM RP TO USER SPACE) R0)+ TPD. (DEC W, BRANCH IF NOT ZERO) ₩ S08+ (PSW BACK TO NORMAL) 777760 84 0 4 HOU. NEXT, (RETURN) BLOCK # 447 **************** **************** (MEMORY MANAGEMENT - U>K) CODE U>K (EU ADRESIEN ADRESIECOUNTI---EI COPIES 'COUNT') (WORDS FROM USER SPACE TO KERNEL SPACE) (W=COUNT) ы. s)+ HOV+ HOU. (RO=KERNEL SPACE ADDRESS) S)+ R O (R1=USER SPACE ADDRESS) S)+ HOV, R1 7777760 ## 300000 # HOV+ (SET PROCESSOR STATUS WORD:) CURRENT=KERNEL, FREV=USER) BEGIN, R1)+ FPD. (FROM USER SPACE TO RP) R0)+ RP)+ MOV, (FROM RP TO KERNEL SPACE) DEC W, LOOP IF NOT ZERO) W SOB, 7777760 80 0 0 HOV+ (CURRENT=KERNEL + PREV=KERNEL) (RETURN) NEXT

¥ S

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 Rochester.

References

- 1. <u>Microcomputers and Memories</u>, Digital Equipment Corporation, Maynard, MA 01754, 1981.
- Lawrence P. Forsley, "FORTH Multitasking in URTH," Proceedings of the 4th West Coast Computer Faire, 1979.
- Michael A. McCourt, <u>PDP-11 FORTH-</u> 79 Implementation Guide, University of Rochester, Laboratory for Laser Energetics, 250 East River Road, Rochester, NY 14623, 1981.
- Rosemary C. Leary and Donald P. McClimans, <u>Omega Alignment System</u> <u>Software Maintenance Manual</u>, University of Rochester, Laboratory for Laser Energetics, 250 East River Road, Rochester, NY 14623, 1981.

FORTH CLASSES

November 16-20 December 7-11 January 11-15

Call: Inner Access (415) 591-8295

A HIGH LEVEL INTERRUPT HANDLER IN FORTH

R. L. Keck and L. P. Forsley Laboratory for Laser Energetics Unversity of Rochester

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.

Introduction

X-ray data from laser-plasma interaction experiments on the GDL 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 CAMAC¹ 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 and read the data from the A/D module.

The above sequence 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 input module and A/D module will generate CAMAC Look At Me's (LAM's) when a signal occurs at their inputs 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 can, in fact, be done and our method for doing this is discussed below.

Implementation

Our system consists of UR FORTH-79 running on a Digital Equipment Corporation LSI-11 microcomputer under DEC's RT-11 operating system. While a complete description of the implementation of this system may be found in the implementation guide², 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 @#<ADDR> ,-(S)], loads the IC with the address of the compilation address of the return from interrupt code [MOV #4HERE+83IC] (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 S 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 task area. The task area contains return 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 SV 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.MACRO is used to generate code which will set the SV and S 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 [52T(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

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.

An Example

The listing for blocks 3 and 4 illustrate how the interrupt handler is used in our acquisition system. A task area (1TASK) 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 device). They share the same task area since only one interrupt service routine can be active at a time.

In block 4, the high level service toutines are defined. RDY.INT is used to thear the A/D module, enable A/D LAM's KCLR XENLAM) and then clear and disable further LAM's from the contact sense nput module, on occurrence of a LAM from the contact sense module. FIRE[collects the A/D data, disables further \Rightarrow D LAM's (XCOLLECT XDISLAM) and activates another task which will print the results (2TASK DISPATCH) on occurrence of a LAM from the A/D module. These gh level routines are installed as the -terrupt service routines for the approstate CAMAC devices with the sequen-RDY.WORD RDY.INT * # SL and TRE.WORD FIRE[. Changing an interrupt service routined with this system requires they defining a new high level handler -ord and installing it as the handler word, e.g., FIRE.WORD FIRE2[will make the -ord FIRE2[the new interrupt service to_time for the A/D module.

Canclusions

We have shown that it is possible to -:te high level interrupt service routines -FORTH. This makes it possible for prorammers unfamiliar with interrupt protramming to easily write interrupt service routines. In addition, the facility with -> ch this system permits changes to be rade to the interrupt handlers makes this => ideal way to handle data acquisition in => rapidly changing experimental environrent.

Acknowledgement

The authors would like to thank travel McCourt for assistance with retails on the internal operation of UR FORTH-79.

L. Keck is a graduate student in Mechanical Engineering at the University of
 Schester. L.P. Forsley is Group Leader
 Computer Systems at the Laboratory
 Laser Energetics, University of
 Schester.

Modular instrument and digital interface system (CAMAC, IEEE STD. 583-1975)

McCourt, Michael, "University of Rochester PDP-11 FORTH-79 Implementation Guide," Release Number 1.0, May 1981, unpublished.

APPENDIX Word Listings

BLOCK (High level FORTH interrupt handler rlk lpf 25-may-81 (<>-<>, restore registers 0-5 \$) : REG.RESTORE.MACRO ASSEMBLER 0 5 DO I RP >+ MOV, -1 +LOOP FORTH # (restore registers, return from interrupt #) CODE RTI RP 52T SV I) HOV, REG.RESTORE.MACRO RTI, FORTH : XEQ.MACRO (<addr of xee word; assembly time>-<> \$) ASSEMBLER S -) SWAP 20 MOV; (push handler word addr on stack) IC HERE 8 + # MOV, (preset the IC) ' EXECUTE P JMP, (Jump to execute)
 COMPILE RTI (pointer to next instruction) FORTH # : REG.SAVE.MACRO $(\langle \rangle - \langle \rangle$, save registers $0 - 5 \pm)$ ASSEMBLER 6 0 DO RP -> I MOV, LOOP FORTH # --> BLOCK 2 ******************************** 25-may-81 (more interrupt stuff rlk) (<slot#><code addr>-<> set camac vector #) : SETUP.INT SWAP 4 # 4000 + DUP ROT SWAP ! 2+ 3400 SWAP ! # : SV.SET.MACRO (<SV loc>-<> set SV for interrupt routines #) ASSEMBLER SV SWAP & MOV, S 14T SV I) MOV, 52T SV I) RP MOV, RP 16T SV I) HOV, FORTH # : CREATE.CAMAC.INT.WORD (<SV loc><slot#>-<>, create int. #) (defin. word. #) <BUILDS 0 , HERE SETUP.INT HERE 2- REG.SAVE.MACRO SWAP SV.SET.MACRO XEQ.MACRO DOES> [COMPILE] INSTALL SWAP ! ; --> BLOCK (Interrupt task area initialization rlk 16SEP81) 20 30 0 47 BLDTASK 1TASK (create @ task area \$) 1TASK TELEAR (initialize task area #) TASK DUP SV DUP ! (delink task from task list *) 1TASK DISPATCH (mark task as active #) (create a ready to fire handler word for CAMAC slot 6 \$) 1TASK & CREATE.CAMAC.INT.WORD RDY.WORD (create a fire time word for the A/D module #) 1TASK XAD CREATE.CAMAC.INT.WORD FIRE.WORD :5 BLOCK 13-apr-81 rlk) (xray interrupt service 40 120 0 47 BLDTASK 2TASK (task area for post fire word #) : RDY.INT (rdw fire int handler #) XCLR XENLAM 6 N 0 A 2 F DROP 24 F F : FIRE! (fire time handler #) XCOLLECT XDISLAM 2TASK DISPATCH # (make RDY.INT the ready to fire *) RDY.WORD RDY.INT (interrupt service routine *) FIRE.WORD FIRE! (make FIRE! the fire time interrupt handler *)

-->

OPTIMIZED DATA STRUCTURES FOR HARDWARE CONTROL

Joseph D. Sawicki Laboratory for Laser Energetics University of Rochester

Abstract

Data structures have been developed to more easily control hardware. A disk driver is used as an example for exploring alternative FORTH data structures and ways of optimizing them. These examples show that FORTH data structures are well suited to minimizing programming time and increasing software efficiency.

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⁻ RX02 mode floppy disk drive. In doing this certain commonly used FORTH tools became useful. This paper will serve to illustrate these tools, and the modifications necessary due to the nature of the project.

Deta 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 concept² in variables. This could be implemented in high level as follows:

It would be used like a variable. Entering 0 VAL<NAME>would define a variable with an initial value of zero. To change 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 eliminating the use of (a) and [with variables (and ' with constants) to access and modify them. The backup driver is no exception to this and in fact offers the opportunity to carry the concept one step further. In the DEC PDP-11 architecture, I/O is memory mapped so that, for instance, the Disk Control Status Register is at location 1771700⁻² and the Data Buffer Register is at location 1771720. One way to communicate with these addresses is to define two constants:

1771700 CONSTANT CSR 1771720 CONSTANT DBR

but then the use of and [becomes necessary. A way 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 address as well as to send data to and from the address. An easy, though by no means optimal, implementation of such a structure is given below.

> : TO (SETS FLAG SO THAT A NUM WILL BE STORED IN A REG.) 1 ZTO 1 ; FROM (SETS FLAG SO THAT A NUM WILL BE GOTTEN FROM A REG.) -1 ZTO 1 ; (TEST BED FOR BEGINING OF RXO2 DRIVER JDS 15JUNG1 : REGISTER (BUILDS , (ABD--C), BUILDS A DATA TYPE CALLED A REGISTER) DOES> (GIVES REGISTER ABD, CONTENTS OR SENDS DATA TO THE REGISTER DEPENDING ON THAT STATUS OF SETO 0 ZTO 0 (GET ADDRESS OF REG AND SETO DUP -1 = IF SMAP 0 SMAP (GET CONTENT) THEN 0 SETO

Once these two structures are implemented it becomes very easy to talk to the disk drive. For example, if a VAL had been defined called IN-TRACK# which contained the track to be read, sending it to the DBR would simply consist of saying IN-TRACK# TO DBR.

1

,

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-DENSITY (<COM.>-<COM.>, SETS THE DENSITY BIT TO 0)
DRIVE/DENSITY 255 BIC TO DRIVE/DENSITY +
DOUBLE-DENSITY (COM.>- COM. -, SETS THE DENSITY BIT TO 1)
DRIVE/DENSITY 256 BIS TO DRIVE/DENSITY +
ODRIVE (<COM.>- COM. -, SETS THE DRIVE BIT TO 0)
DRIVE/DENSITY 16 BIC TO DPIVE/DENSITY +
IDRIVE (<COM.>-COM. -, SET THE DRIVE BIT TO 1)
DRIVE/DENSITY 16 BIS TO DRIVE/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 point. 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, as shown:

: (SET-DEN) 110 DRIVE/DENSITY OR TO CSR ;

The second approach would be to again use a defining word:

: DISK-COMMAND BUILDS (<CON --> TAKES THE CON FOR A DISK OP,) DOES (GET COM AND DRIVE DEN INFO DR, AND SEND) P DRIVE/DENSITY OR TO CSR ;

110 DISK-COMMAND (SET-DEN) (USED TO FORMAT DISKS SING OR D DEN)

Optimization

As usual we have a classic FOR [H space-time tradeoff. The second approach executes somewhat slower (see figure 1) because the constant needs to be fetched, but whereas the first approach 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:

: VAL <RUILUS (.+)--->, TAKES THE INITIAL VALUE OFF THE STACK) ;CODE (<+>-> OR <>--(+), GETS VALUE OR STORES VALUE) ZTO F TST, (SEE IF ZTO POSITIVE) GT IF, WPARAN W I) S)+ MOV, (STORE VALUE) ZTO F 0 + MOV, (ZERO OUT ZTO FLAG / ELSE, S -) WPARAN W I) MOV, (FETCH VALUE OF VAL) THEN, MEXT,

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 I) 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:

: REG <BUILDS (BUILDS A DATA TYPE CALLED A REGISTER)
;CODE (`\$.--.>,<>-..>, GETS ADD, VALUE OR STORES VAL)
XTO F TST, (CHECK IF XTO IS POS NEG OR ZERO)
GT IF,
WFARAM W @I) S)+ MOV, (STORE VALUE IN REG)
ELSE,
LT IF,
S -> WFARAM W @I) MOV, (GET VALUE)
ELSE,
S -> WPARAM W I) MOV, (FUT T.O.S.)
THEN,
THEN,
THEN,
THEN,
-->

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 transfer request bit to be asserted in the CSR.

: TR.WAIT (WAITS FOR THE DATA TRANSFER BIT TO BE SET) BEGIN 2000 FROM CSR AND END ; DONE.WAIT (WAITS FOR THE DONE BIT TO BE ASSERTED) BEGIN 400 FROM CSR AND END ; The disk command as shown before was called (SET-DEN). After receiving this command the disk controller waits for a "key" byte (1110, the letter I in ASCII) to be sent to the DBR, therefore the entire command is coded as shown:

```
: FORMAT-DISA ( <>-<>, SETS THE DENSITY OF A DISK )
(SET-DEN) TR.WAIT
1110 TO DBR ( SEND *KEY* BYTE )
DDNE.WAIT ;
DDNE.WAIT ;
```

To format the disk in the drive one double density one would enter IDRIVE DOUBLE-DENSITY FORMAT-DISK; to format the disk in drive zero single density one would enter ODRIVE SINGLE-DENSITY FORMAT-DISK.

Timing

.

To show the effects of the different approaches timing tests were run. The first contrasts the difference between the two types of disk commands. In all tests the action was placed inside a double loop like:

: TEST 10 0 DO 30000 0 DO LOOP LOOP ;

This routine took 23 seconds which was then subtracted 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. These are the results on a DEC LSI 11/2:

To Send Disk Command

Colon definition	.23 msec.	
Defining word	.28 msec.	

Then a high level VAL was compared to a coded VAL and a VAR:

	fetching (msec)	storing (msec)
high level VAL	. 237	. 39
coded VAL	. 067	.11
VAR	.083	. 093

Summery

This paper not only showed the usefulness of certain techniques in FORTH but also illustrates some general properties of the language. The first of these is the ease of implementation of new data structures. Through the use of BUILDS ... DOES or BUILDS ... ;CODE one can first build the structure to suit the needs of the application and then imbed in the executable code necessary operations for the structure. Also a structure can easily be given variable execution as in the case of VAL and REG. Another important benefit of FORTH is the ease of optimization of the word by the use of assembly code. Changing the VAL and REG words to ;CODE took less then a half hour.

Acknowledgements

I would like to thank Mike McCourt, Bob Keck, Lawrence Forsley and Peter Paulson for their help in getting the hardware running and for comments on the software.

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-AC08-80DP40124.

J. Sawicki is an undergraduate with the Electrical Engineering Department of the College of Engineering at the University of Rochester. He is a DJ in his spare time.

^{1.} DEC and PDP-11 are trademarks

^{2.} The TO concept by Paul Bartholdi FORTH DIMENSIONS Vol. I No. 4 and Vol. I No. 5.

^{3.} Where an O suffix indicates octal

THE STRING STACK

Michael McCourt Laboratory for Laser Energetics University of Rochester

Richard A. Marisa Production Automation Project University of Rochester

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.

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<NAME>places

seginning address><maximum 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

<1><NAME>

places <address of the i-th string> <maxlen>

on the parameter stack. Note that (number of elements) x (maxlen) bytes will be allocated to hold the string array.

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 can 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 .SS is used to print the top element of the string stack,

***************** 96

(STRING STACK--FIXED LENGTH STRING COMPARISON LAR 19-SEP-79) ; SS ; (NOTE: PARAM ORDER NOW $\langle ADR \rangle \langle LEN \rangle$ NAM 11-JUN-BG ; (LADD A; AUD B; LEN],---LAUDA; ADDB; = OR + OR - J) (COMPARES CHARS. IN STRINGS A & B PARIWISE; RETURNS O IF (STRINGS AKE =, + IF A>B; - IF A>B)

- \$ S?FDD 0 SWAP 0 DO DROP OVER C0 OVER C0 ROT 1+ ROT 1+ ROT DUP 0= NOT IF LEAVE THEN LOOP; (LADD A, ADD B, LEN3---L= OR + OR -3; SAME AS S?FDO) (EXCEPT ADDRESSES NOT RETURNED)
- : STE STEDO ROT ROT 20RDP (I ADD A, LENJ--(= OK + DR -], COMPARES STRING A TO) (A STRING OF BLANKS--RETURNS O IF TWO ARE EQUAL) (STR 0 SWAF O DO DROP DUP CP BL - SWAF 1+ SWAF DUP OK) IF LEAVE THEN LOOP SWAP DROP ; ---

97 ****************

***************** 98

- STRING STACK WORDS LAR 19-SEP-79)
- (STRING STACK WORDS LAR 19-SEP-79)
 O SVAR SSD O SVAR SSM O SVAR SST
 SSTOP SST @ ; ; SSTOP SST ! ;
 SSORG SSD @ ; ; SSMAX SSM @ ;
 (EFROM, TO, LEN]---[] CHECKS FOR STACK BOUNDARIES)
 SOUCHECK OVER SSORG U:
 IF SSMAX SSTOP ! 14T TABORT THEN ;
 (LADD]---[] INSURES THAT ADDRESS POINTS TO STRING)
 SSUED DUP C@ + SSMAX U>=
 IF SSMAX SSTOP ! 13T TABORT THEN ;
 (ADD OF TOF STRING]--LAD OF NEXT STRING DOWN])
 SSDOWN DUP C@ 1+ ;
 (EADD]---[] PUSHES STRING AT ADDR. TO TOS)
 SSPUSH DUP C@ 1+ SSTOP OVER DUP SSTOP! SWAP RMOVE ;

- -->

99 ****************

STRING STACK WORDS LAR 19-SEP-79) "DROP ([]---[] REMOVES TOP STRING FROM STACK *) SSTOP SSVER SSDOWN SSTOP'; "LEN ([]---[] RETURN LEN OF TOS STRING *) SSTOP SSVER C0; "LOC ([]---[] RETURN ADDR OF TOS STRING *) SSTOP 1+; 'LOC SSTOP 1+ ; • ([]---[] COPY TOS STRING #) *DUP ([]---[] COPY TUS SIKING */ SSTOP SSVER SSPUSH ; SWAP ([]---[] EXCHANGE TOP 2 STRINGS *) SSTOP DUP SSDOWN DUF SSPUSH SSDOWN SSTOP SWAP SSTOP! SWAP SSPUSH SSPUSH ; ROT ([]--[] ROTATE TOP THREE STRINGS ABC->BCA *) SSTOP DUP SSDOWN DUP SSDOWN DUP SSPUSH SSDOWN SSTOP SWAP SSTOP! SWAP SSPUSH SWAP SSPUSH SSPUSH ; --> DUP : **************** 100

(STRING STACK WORDS

MAH 13-JUN-80)

-->

([]---[] PUSH 2ND STRING DOWN ONTO TOS \$) : "OUFR :

"DROP "DROP ; ' *ZOVER ([]---[] PUSH 3RD AND 4TH TO TOS #) SSTOP SSDOWN SSDOWN DUF SSDOWN SSVER SSPUSH SSPUSH ; ' ZSWAP ([]---[] EXCHANGE IST # 2ND WITH 3RD AND 4TH #) DUF SSDOWN SSDOWN SSTOP' SSPUSH SSPUSH ; BUP SSDOWN SSDOWN SSTOP' SSPUSH SSPUSH ; DUF SSPUSH ; ' DROP SSPUSH ;

-->

removing the top element in the process. For example,

OK " STACK THIS STRING " <CR>

OK

.SS <CR>

STACK THIS STRING OK

Notice that the functions .SS and . are similar. Several other functions operate on the string stack in a manner analogous to words which operate on the parameter stack. These are:

(STRING STACK WORDS CON	T'D M	AM 13-JUN-BO)
:			ADDR. 1 DROP TOS 1)
	SSTOP "DROP SWAP OVER Rot swap Rmove ;		
	([STRING][] STORE (THERE TO THE TOSS -	S STRING IN PAD	THEN NOVES IT FROM)
ŧ	X" 420 WORD 0 "P ;		EXECUTION TIME /
;	S' R> DUP O 'P DUP CP IF 1+ ELSE 2+ THEN +	DUP 2 NOD	
	([STRING][] STORE	S STRING AT TOP	OF DICT. STACK)
:	(DURING COMPILATION C' COMPILE \$' 420 WORD	CE DUP 2 HOD	
•	IF 1+ ELSE 2+ THEN AL STATE & IF C' ELSE X		
•	STATE & TE C ELSE A	INCR 7 INF	>

102

(STRING STACK WORDS CONT'D HAM 18-MAR-81) : .SS ([]---[] TYPE OUT STRING AT TOSS *) SSTOP SSVER *DROP COUNT TYPE ; : ', (<>-<>, PUT STRING IN DICTIONARY, MAKE EVEN LENGTH) 420 WORD COUNT DUP HERE SWAP 1+ -2 AND ALLOT SWAP CHOVE ;

(SOME FIXED LENGTH STRING DEFINITIONS)
 (IADDR,MAX LEN]---[] PUSH STRING AT ADDR TO TOSS)
: "@F DUP SSTOP OVER - 1- SSTOP! SSTOP C' SSTOP
 1+ SWAP CHOVE ;
 (LADDR,MAX LEN]--[] COPY CHARS ONLY FROM TOSS TO ADDR)
: "!F 2DUP BLANK "LEN MIN SSTOP 1+ ROT ROT CHOVE "DROP ;

103

101

-->

'n

2

9

s: 5 11 K. K.

7

9

5.2

7w 201

S.: 8C

-

24

is:

æ **1**07

Se

というない

940 1-11

29

Ē

WORD	FUNCTION	BEFORE	AFTER
"DUP	copies top of stack	BA	BAA
"SWAP	reverses top two		
	strings on the stack	BA	A B
"DROP	removes top of stack	BA	B
"OVER	copes 2nd string onto top	BA	ВАВ
"ROT	moves 3rd string to top	СВА	BAC
"2DUP	copies top 2 strings	ВА	BABA
"2DROP	removes top 2 strings	СВА	С
"2SWAP	reverses 1 & 2 with 3 & 4	DCBA	BADC
"20VER	copies 3 & 4 to top	DCBA	DCBADC
"+	string addition (catenation)	BA	BA

String Relationals

Just as the parameter stack relational operators remove their arguments from the parameter stack, the following string stack relational operators remove their arguments from the string stack. The logical result of the string relation is placed on the parameter stack. The available relationals are:

> H.... "<= "<> ">=

"<

String Variable Storage and Retrieval

The string store word, "[, places the top of the string stack in the string variable described by the parameter stack, 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 "! <CR>

OK

```
MYSTRING "@ MYSTRING "@ "+ .SS <CR> string text string text
```

OK

T-oking the name of the string variable CSTRING in the preceding example aced (address) (maxien) on the parameter stack. String store and string retrieve check the maximum and current ergth of the string variable when moving Dring data.

When it is required to move fields of ied length which do not contain an impedded current length in the first byte, ied length string store and retrieved words may be used. The syntax is:

<address> <length> "!F

<address> <length> "@F

Scring Functions

"LEN returns on the parameter stack, relength of the string on top of the ing stack. The string remains on the ing stack. The address of the first byte the string (one byte after the length e d is found by executing "LOC.

replaces the top of the string stack with a substring of length (length), beginning + in the specified character of the ingunal string. For example,

ЭK

" abcde" 2 3 "substr .SS cd OK

The "INDEX function searches for the rst occurrence of the string in the woond string. If an occurrence is found, d offset is returned on the parameter mack. If an occurrence is not found, -1 is mourned. The top of the string stack is thosed.

Scring Stack Errors

We errors are reported by the string Fack package: string stack underflow and Fellow. As stated previously 200 bytes Fellow. As stated previously 200 bytes Fellow. As stated overflows are generstate. If repeated overflows are generfor more space can be allocated for the Fing stack by changing the parameter Facts to "INIT in the string stack that, String stack initialization is the stit function performed when the string Fact library is loaded.

iummary

This was the first major software :=:<age transported throughout the :=::<age transported throughout the :=::<age transported throughout the :=::<age transport of the solution of the sol

104 *************** (STRING STACK WORDS CONT'D LAR 19-SEP-79)
 ([]---[-1 OR OFFSET] SEARCHES FOR 1ST OCCURENCE OF)
 (TOP STR. IN 2ND STR.---IF FOUND OFFSET IS RETURNED)
 (ON PARAM STACK ELSE -1 IS RETURNED. TOSS IS POFPED .)
'INDEX -1 SSTOP DUP C@ 0
'IF DUP SSDOWN SSVER DUP C@ ROT 1+ C@ ROT 1+ SWAP
ROT 0 'INDEXDO
 ELSE 0 ROT ROT THEN 2DROP 'DROP ;
 ([]---[] COMPARE % DROP 'DROP 'DROP ;
 ([]---[]/F] LOGICAL =, TESTS TOP 2 STRINGS)
'* "Y O=;
 ([]---[T/F] LOGICAL LESS THAM TESTS TOP 2 STEINGE \ ; *= *? 0= ; ([]---[T/F] LOGICAL LESS THAN TESTS TOP 2 STRINGS) : *< *? 0 > ; ---**************** 105 */ (L)---LT/FJ TESTS TOP 2 STRINGS FOR > *) (STRING STACK WORDS CONT'D = ([]---[T/F] TESTS TOP 2 STRINGS FOR <= #) • (= ([]---[T/F] TESTS TOP 2 STRINGS FOR >= #) : •S= "INIT (<*CHARS TO ALLOCATE FOR SS>-<>> INIT SS INTO DICT) 1 ?* HERE SSD ! ALLOT HERE 2- DUP SSM ! SST ! ; 2001 "INIT (ALLOCATE 200 CHARS FOR STRING STACK) ***************** 106 (STRING VARIABLE AND STRING ARRAY MAM 13-JUN-80)
 ((MAX LEN)---[] ALLOTS SPACE IN DICT FOR MAX LEN AND)
 (MAX & OF CHARS.)
 STRING-SPACE DUP , 0 , 2/ DP+! ;
 (IMAX LEN] STRING (NAME> --- BUILDS A STRING VARIABLE)
 (WHEN (NAME> IS EXECUTED THE BYTE ADDR. OF THE STRING)
 (START AND LENGTH ARE LEFT ON THE STACK)
 STRING-VAR (BUILDS STRING-SPACE
 ;CODE S -) W MOV, (PUSH PARAM ADDR)
 S) 4 & ADD, (PUSH PARAM ADDR)
 S -) 2 W I) MOV, (PUSH PAX LENGTH)
 NEXT. NEXT --> **************** 107 (STRING ARRAY ROUTINE MAH 13-JUN-80) SIRING AKKAT KUUIINE () ()STRING (C4 OF ELEMENTS, MAX LEN] ---(NAME> \$) (BUILDS SWAP DUP , (BUILD NEADER, STORE 4 OF STRINGS) O DO DUP STRING-SPACE (ALLOT DIC SPACE, STORE MAX LEN) LOOP DROP

DOES> 2+ DUP @ ROT ROT 3 PICK (ADDR OF 1ST ELEMENT) DUP 2 MOD IF 3 + ELSE 4 + TMEN (1+ TO MAX LEN IF ODD) (2+ IF EVEN, 2+ FOR MAXLEN) ROT \$ 2+ + SWAP ; (STRING ADDR + ELEMENT OFFSET) (RETURNS COUNT AND ADDR)

-->

iS

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 word processing system. The package consists almost entirely of its original code written in 1977 by Mike Williams, of the University Computing Center. The major change has been the addition of comments.

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 8080; and two undergraduates who worked for Lawrence Forsley, Lynn Raymond and Dan 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-AC08-80DP40124.

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.

HELP WANTED

Associate Systems Manager, Pulmonary Computer Systems

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

NEW PRODUCTS

FORTH Application Modules Diskette

The diskette of FORTH application moduels, 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 definitons, and conversational programs. One hundred screens of software and one hundred screens of instructional documentation are supplied on the diskette. Every screen is in exemplary FORTH programming style.

The FORTH screens, written by Scott Pickett, may be used with Timin FORTH or other fig-FORTH. The price for the diskette of FORTH application modules is \$75 (if other than 8" standard disk, 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.

INNER ACCESS FORTH SOFTWARE AND 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" single density diskette only. #20011-01 (\$80)

Reference Manual for PDP-11 fig-FORTH above. #20011-99 (\$20)

Fig-FORTH compiler/interpreter for CP/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, with 5.25" diskettes in 128 byte, 18 sector/track format and 8" 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" diskette with source in 8080 assembler #20080-88 (\$80).

Released on two 5.25" diskettes with source in Z80 assembler #20080-Z5 (\$80).

Released on one 8" diskette with source in Z80 assembler #@0080-Z8 (\$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 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 Model II under CP/M Northstar Horizon Prolog Z80

Released on two 5.25" diskettes #20100-85 (\$1,000).

Released on one 8" diskette #20100-88 (\$1,000).

Complete Zilog (AMD) Z8002 develoment system that can be run under CP/M or CROMEMCO CDOS. System includes a METAFORTH Cross-Compiler which produces a Z8002 fig-FORTH compiler/interpreter for the Zilog Z8000 Development Module. Package includes a Z8002 assembler, a Tektronix download program and a number of utilities.

Released on two 5.25" diskettes #29102-85 (\$4,000).

Released on one 8" diskette #29102-88 (\$4,000).

Zilog Z8002 Develoment Module fig-FORTH ROM set. Contains fig-FORTH with Z8002 assembler and editor in 4 (2716) PROMS. #38002-00 (\$850).

For orders and further information, contact:

INNER ACCESS CORPORATION Software Division Box 888 Belmont, CA 94002 (415) 591-8295

ANNOUNCEMENTS

Sym-FORTH Newsletter now available, contact: Saturn Software Ltd., PO Box 397, New Westminister, British Columbia, V3L 4Y7, CANADA. 2

1

:

3

1

1

COMPLEX ANALYSIS IN FORTH

Alfred Clark, Jr. Department of Mechanical Engineering University of Rochester

During 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, I found the transition to FORTH somewhat difficult and even painful at times. Originally, I 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. Among 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 is regarded as an ordered pair, and is pushed on the stack by typing 3.5 7.2. With this convention established, it is easy

to define all of the important stack manipulations such as ZDROP, ZDUP, ZOVER, ZROT, and ZSWAP, which perform exactly like their integer and floating point counterparts. The basic load and store operators, Z@ 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 ARGZ (radians). Most of these are quite simple to define. IMZ, for example, is just : IMZ FSWAP FDROP; where FSWAP and FDROP are floating point stack operations. As another example, consider 1/Z defined by : 1/Z ZDUP /2/2 FROT FOVER F/

FROT FROT F/ COLU ;

For ARGZ it is very important to establish 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 rectangular 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

: POLAR ZOUP /Z/ FROT FROT ARGZ ;

A similar word, DPOLAR, leaves the argument in degrees. For conversion from polar to rectangular, we have RECT (angle in radians)

: RECT FOVER FOVER COS F* FROT FROT SIN P* ;

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 $Z \in ROTZ$ rotates Z by F radians and leaves the result on the stack. The definition is

: ROTZ FROT FROT POLAR FRUT F+ RECT ; .

There are several different useful formats for complex output. (*'y system has 8 different formats, which is handy but a little extreme.) The word Z. prints the number as an ordered pair -- 3.5 7.2, for example. The conventional mathematical notations is obtained by Zl. -- (3.5) + (7.2)l. Words to print in polar form are also useful. For example, ZP. is defined so that the sequence 3.5 7.2 ZP. gives

MOD = 8.00562303 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

: Z. FSWAP 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

: Z+ FROT P+ FROT FROT F+ FSWAP ;

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 FISHAP EXP FOUP FROT FOUP COS FROT F*

FROT F* FROT FROT SIN F*

Other useful functions such as ZSIN, ZCOS, ZTAN, ZSINH, ZCOSH, and ZTANH are defined similarly.

Of the multi-valued functions, the most useful are the square root ZSGR, the logarithm ZLOG, and the power Z**. As an example of the definitions, consider the principal value of the square root:

: ZSOR FOLAR 2. F/ FSWAP SOR FSWAP RECT ;

The basic words described above 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 any previously defined complex function, and produce a graph showing the curve and its image under the transformation. This tool allows students (and the instructor) to improve their understanding of the geometry of complex functions.

Notes on Implementation

The code described above runs on the author's 48K 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 2K. The conformal mapping code compiles to about 1K 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 Foreign

ORDER NOW!

A FORTH BASED MICRO-SIZED MICRO ASSEMBLER

Gregory E. Cholmondeley Laboratory for Laser Energetics University of Rochester

Abstract

The FORTH programming language can be used to implement a very small 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 microcoding technique. This paper also serves to illustrate several of the many programming features found in FORTH.

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 code 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 drawback: they are large programs. The micro assembler presented here is based heavily upon the Signetics¹ 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.

Usage

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 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. FIELD 8 WIDTH 2 Define field B as the next 2 bit positions, having a default value of 0.

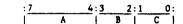
defined. A simple example of such a

FIELD C WIDTH 2 DEFAULT 1 Define field C as the 2 least significant bits, having a default value of 1.

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 and may not be used to define other fields.

While the preceding example is rather trivial an instruction definition may become quite complex. It is, for instance, possible to define multiple formats 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 allowing an unlimited nesting of fields, formats and sub-fields. Figures (1) and (2) should clarify this concept.

This part of the micro assembler has error checking capabilities which prevent unintentional overwriting of fields. For example, if field EE of figure (1) is filled, then fields B8, 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 AA, CC, FF and HH will be defaulted.

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		
FORMAT. END		

FORMAT FIELD GG WIDTH 16 DEFAULT 65535 FORMAT.END FIELD HH WIDTH 8 DEFAULT 255 END.INSTRUCTION

Figure (1) : Sample Instructon Definition

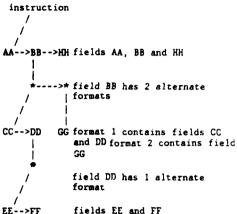


Figure (2) : Structure of Figure (1)

The programming phase of the micro assembler is where the actual microcoding takes place. An instruction is created by typing the name of a field followed by a number or expression 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 instruction 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 sets one field at a time equal to a zero.

PROGRAM 1EXAMPLE WIDTH 32

=

3

ORG 512

AA BB	-	Ş S		
CC	0	\$		
DD EE		\$ \$		
FF GG	-	\$ \$		
HCH	-	\$		
END. PROGRAM				

Figure (3) : Sample Program

::::::::::::::::::::::::::::::::::::::	11111111111111 11111111111111 11111111	BB used CC used DD used EE used FF used GG used	BB & HH defaulted AA & HH defaulted AA, DD & HH defaulted AA, CC & HH defaulted AA, CC, FF & HH defaulted AA, CC, EE & HH defaulted AA & HH defaulted AA & HH defaulted
	111111100000000		AA & BB defaulted

Figure (4) : Sample Output

While automatic field alignment is in tself a vast improvement over hand coding, there are a few other tools available to the programmer which make microcoding even easier. A "(." denotes a comment allowing anything up to and reluding a ".)" to be ignored. Typing ORG and a number or an expression will set the position counter (LC) to that value. yping _____SET <new variable name>

TO <number or expression>

will declare and initialize a variable, while :voina EQU <old variable name>

WITH <number or expression>

will store a new value into a previously seclared variable. These variables return their value when they are typed (similar to a constant in FORTH) and can be used in expressions at any time and in any phase :' the micro assembler.

One of the most versatile tools available in this micro assembler is the CROP function. Microps are user- \mathfrak{T} finable functions designed to eliminate a large part of the repetitious programming associated with microcoding. For example there may be times when several leids will always take on constant or relative values. Rather than cluttering the program by having to set all of these fields every time, a microp can be written to do this automatically. A program written using well named microps can in turn be quite a bit easier to read and understand than one which merely sets the fields.

The definition of a microp requires a inique name and a set of commands which will be executed whenever its name is called. Any FORTH programmer will soon ealize that a microp definition is nothing ther than a colon definition, thus allowng the full power of FORTH to be easily accessed directly from the micro assempler[An example of a simple microp that sets a few fields to zero would be:

MICROP EX1 (. set fields CC, FF, CC 0 and HH to 0 .) FF 0 HH O END.MICROP

An example of this microp in use would be found in the programming phase and might look like:

AA 7 HH (LC) \$ AA 8 EX1 \$

NOTE: LC in the preceding example is a variable, the "(" and ")" are required for its proper execution. They do not denote a comment in the MICRO vocabulary context. This is also true when building microps. In the MICRO vocabulary comments are delimited by "(." and ".)".

Being simple colon definitions, microps can do internal testing, looping and every-thing else offered in FORTH. Microps can expect parameters on the stack 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 assembler is through the use of labels. This feature 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 label is found immediately preceding a new instruction word (or in other words; immediately following a "\$") the current value of the location counter (LC) is stored as the value of the label. Multiple labels may be used to represent the same line of code. When a label is used inside an instruction definition after its value has been set, it will be treated as any other variable. If the label has not been set to a value (i.e., forward referencing) a zero will be returned and all information necessary to resolve the reference will be stored in memory for the second pass. During the second pass the micro assembler will shift the correct value(s) of the label(s) into the proper place(s) and then add the resulting number to the rest of the word. This allows labels to be referenced more than once in a single instruction. It also allows addition and subtraction of other non-label expressions to labels (i.e., AA (1LABEL + 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.

(. <exprl> ?GT <expr2> -- tests if expr1 is > expr2 .) MICROP ?GT GET# > IF AA O BB O CC O ELSE HH (LC) THEN END. MICROP

This could be used like:

AA 19 \$ <variable.name> ?GT 1024 \$

Finally, microps have macro capabilities in that they can be nested and may even create several lines of code in one call (as may be needed in a test and branch, or jump substitute routine).

```
MICROP EX3
 LC 100 >
   IF EX1 $
      LC ?GT 1000 $
   ELSE AA 0 $
        CCOHHOS
   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 special 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:

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!

Implementing Techniques

The first problem that I addressed was how to align the fields in an instruction word definition. For words that are 32 or fewer bits wide the solution is simple, merely do logical shifting and ORing. Since 32 bits is a rather stringent limit on the word width, I have kept the same basic strategy but have defined a set of functions 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:

- 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 stack. 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.
- E.DROP This function drops an extended precision number from the top of the stack.
- 4. ESL The ESL function performs a logical shift to the left on an extended precision number. It expects the extended precision number and the number of shifts on the stack and returns the shifted number.
- EOR This takes two extended precision numbers off of the stack, logically ORs them together and returns the resulting number.
- EXOR This executes an exclusive OR operation between two extended precision numbers. It expects two extended precision numbers and returns the result.
- ECOM ECOM does a 1's complement of the given extended precision 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_{+} .

SXXXXXXXXXXXXXXXX BLOCK 160 ************** GEC 15-JUL-81 / (algebraic notation GET# ([<>--<input expression's value>] / 32 WORD NUMBER NOT (set next input char/num) IF R> R> SWAP >R >R THEN # (if char then treat as '(' : (. [COMPILE] (# IMMEDIATE (define (. as comment delimiter) MICRO DEFINITIONS VOCABULARY NICRO Hilton Derinflions (. [<1)=-<01 + 02>] redefine + .) (. [<1)=-<01 - 02>] redefine - .) (. [<1)=-<01 + 02>] redefine # ./ (. [<1)=-<01 + 02>] redefine # ./ (. [<1=-<] end expression .) (. [<1=-<] start expression ./</pre> : + GET# + ; - GET# - ; : # GET# # ; / GET& / #) R> R> SWAP >R >R # FORTH DEFINITIONS *************** BLOCK **************** 161 (value and flipflop types GEC O VAR ZTO (flag) : TO 1 ZTO ! ; 10-JUN-81) : VAL (returns value of variable [not address]) <BUILDS + DDES> ZTO_P IF ! 0 ZTO ! (store value) (store value : i push value : ELSE P : FLIPFLOP (returns 0/1 and stores 1/0) <BUILDS 0 , ([<>--<>] initialize F.F) (BULLDS 0 / PETUFINS 0/1 and (BULLDS 0 / DOES> XTO 0 IF ! 0 XTO ! ELSE DUP 0 BUP NOT ROT ! THEN ; ([<1/0>--<>] set F.F. ([<>--<1/0>] flip F.F. ----162 *************** (variable definitions 0 VAL CUR.ADDR 0 VAL C.FIELD 0 VAL C.FORN 0 VAL C.INSTR 0 VAL F.LENGTH 0 VAL F.LENGTH 0 VAL LC 0 VAL LC 0 VAL LETELD 19-110-81 **BFC** (current address current field current format (current format) (current format) (current instruction word) (field length ; (field rosition) (location counter) (instruction width) (last field) (last format : VAL INSTRUID VAL L.FIELD VAL L.FORM VAL L.INSTR VAL MEM VAL NEW.WORD VAL OFFSET (last field) (last format) (last instruction) (current memory addr for print routines ; (flas set at start of new instr. word ; (offset of shift (used in ESL)) 8 Ā 163 ******************* (variable definitions - 2 GEC 19-JUN-81)
0 VAL OVFL6 (overflow flas)
0 VAL PLACE (addr of temp storade in extended operations)
0 VAL PRECISION (Precision of word in 16 bit units)
0 VAL TST.FLAG (flas used in error checkins and defaultins)
0 VAL TSHIFT (intermediate number of shifts (ESL))
0 VAL ZFLAG (default phase {0. use/1. set/2. initialize) }
0 VAL ZFLAG (default phase {0. use/1. set/2. initialize) }
0 VAL ZFLAG (field F.F. for error checkins 1 defaultins;
0 VAL ZPRINT.FORMAT (addr of output format code ;
FLIPFLOP FLD.FF (field F.F. for error checkins 1 defaultins;
0 XEQ BROTMER (brother of current field/format)
0 XEQ PARENT (parent of C.FIELD) (C.FIELD) (C.FIELD) (uncle of C.FIELD) O XED SELF --> **************** 164 (extended precision functions GEC 12-JUN-81)

: E.FILL (<● len>-<● 0 ... 0> puts 0's in high order places)
PRECISION SWAP 2DUP > IF D0 0 LOOP ELSE 2DKOP THEN ;

: EDROP (<low-order ... hish-order>-<> drops ext.precision #)
PRECISION 0 D0 DROP LOOP # ___>

?: 3:

2

Ŧ

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 shift 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 list of fields as its children. Each field has a list of formats or a zero for its children. 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 brother. Each node also contains a flag denoting whether it is a valid field or not, a value corresponding to its starting position 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 is 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 flag is set and this is not a new line of microcode, then this field is not defined in the same instruction 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 field will be set along with the flag of its parent, and its parent, continuing up to the root. When a "\$" is encountered, the tree is traversed in the same manner but from the root down and all flags are reset. At the same time any unused brothers of the lowest level fields used will be assigned their default values.

INSTRUCTION FORMAT FIELD

INSTRUCTION FORMAT FIELD

² arent	0	field	format
Frother	0	format	field
.sed Flag	0/1	0/1	0/1
Inild	field	field	format
Field Star	ting Positio	on	11
Field Leng	th		11
lefault Va	lue		1
or Zeros			11
			II

<pre>(extended prec. functions = 2 GEC 12-JUN-81) ; ESL (<low-ord #-shifts="" high-ord="">-<low-ord high.ord=""> shifts #-shifts to left (drops high ov & shifts in 0's } 0 TO OVFLG HERE PRECISION 2 # + DUP TO PLACE HERE DO 0 I ! 2 +LOOP 0 FRECISION 1 = 2# DO I TO OFFSET DUP TO TSHIFT SWAP</low-ord></low-ord></pre>
BEGIN TSHIFT 16 >= (for byte from high to low do)
OFFSET 2 + TO OFFSET TSWIFT 16 - TO TSWIFT
I TO OVFLG (#-shift < 16 (set overflow flag) ELSE (#-shift < 16 (shift normally) DUP TSHIFT <-L OFFSET HERE + DUP @ ROT OR SWAP ! >

(extended prec. functions - 3 BEC 12-JUN-81)
OFFSET 2 + HERE + BUP 0 (handles \$s that are split) ROT 16 TSHIFT>L OR SWAF ! (into 2 bytes by shift) THEN OVFLG NOT 0 TO OVFLG END
-2 ŦĹŎŎ₽ DRŎ₽ PLACE HERE BO I Ø 2 +LOOP \$ (fetch \$ from temp workspace)
<pre>; ?PRECISION ([# of bits][# of 16-bit words]) 0 17 M/MOD DROP SWAF DROP 1+ ;</pre>
<pre>: EGET (I<addr of="" variable=""><ext.pre.#>1) DUP PRECISION 1 + 2# + DO I @ -2 +LOOP +</ext.pre.#></addr></pre>
,
######################################
<pre>(extended prec. functions = 4 GEC 15-JUN-81) : EDR (<ext.pre.\$ -<ext.pre.\$="" ext.pre.\$=""> UR 2 ext.pre.\$>) HERE PRECISION 2# + 1 - DUP TO PLACE HERE DO 0 I ' 2 +LOOP 1 PRECISION 1 + PRECISION I - + PICK PRECISION 1 + PICK OR -1 +LOOP HERE PLACE DO 1 ' -2 +LOOP PRECISION 2# 0 DO DROP LOOP PLACE HERE DO 1 P 2 +LOOP;</ext.pre.\$></pre>
<pre>: ECOM ([<ext.#><not ext.#="">] one complements ext.pre.#) HERE PRECISION 2# + 1- DUP TO FLACE HERE SWAP DO I ! -2 +LOOP PLACE HERE DO I @ COM 2 +LOOP ;</not></ext.#></pre>
: ERROR.FUNCT .* ERROR CODE: * . CR 7>

<pre>(extended prec. functions - 5 GEC 15-JUN-B1) : EXOR (<ext.pre.@ ext.pre.@=""><<ext.pre.@> OR 2 ext.pre @s) HERE PRECISION 2# + 1 - DUP TO FLACE HERE DO 0 I : 2 +LOOP 1 PRECISION + PRECISION I - + PICK PRECISION 1 + PICK XOR -1 +LOOP HERE FLACE DO I ' -2 +LOOP PRECISION 2# 0 DO DROP LOOP PLACE HERE DO I @ 2 +LOOP ;></ext.pre.@></ext.pre.@></pre>
######################################
(offsets in field structure GEC 3-JUL-81) : OFF.VAL
+ XTO @ IF ! O XTO ! ELSE DUP O<> IF @ THEN THEN F
: TPARENT O OFF.VAL ; : TBROTHER 2 OFF.VAL ; : TFLAG 4 OFF.VAL ; : TCHILD 6 OFF.VAL ; : TANCESTOR TPARENT TPARENT ; : TINSTRUCTION.WIDTH 8 OFF.VAL ; (INSTRUCTION) : TFIELD.START C.FIELD 8 OFF.VAL ; (FIELD) : TFIELD.LENGTH C.FIELD 10 OFF.VAL ; (FIELD) : TDEFAULT C.FIELD 12 + ; (FIELD) : NEW.SON DUP TO ILD DUP ROT AND IF 0 SWAF BEGIN DUP TBROTHER ROT DROP DUP NOT END DROP ELSE DROP 0 THEN TO BROTHER ;>

With the structures 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.

General	First	Pass	Structure	for
Microcode	Progra	ms		

	1 1 1	Forth
	!!	
Forth		Name
Header	<u>***</u>	Link
	<u>*</u>	Description
Program	*	 Instruction Word Width
Header	0 1	
	*	Address of Label
	***	- Field (ie. # of shifts)
	: 1	
Complete	1 : 1	
First Pass	i * i	Address of Label
Data For	*	- Field
One		11010
Instruction	 	Outout Formet
	I	- Output Format
Word	*	- LC
	11	Instruction
	11	Word
	*	- Address of Label
	*	- Field
	i <u> </u>	
	i : i	
		Instruction
		Word
	!!	
		End of Program

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 objectives 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 functions to accept numbers and expressions from the input buffer as well as from the parameter stack.

This method uses the return stack via a function GET# which accepts 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 checking whether a set of parentheses is properly closed. However, a variable could be used which would be incremented

170 *************** (headers of fields & formats GEC ; ?NAME DUP 0<> IF CFA TNAME ELSE DROP THEN ; ; IGNORE 32 WORD DROP ; 3-JUL-81) HEADER (creates ist 4 fields in FIELD and FORMAT) O TO UNCLE HERE TO SELF BROTHER 0(> IF SELF BROTHER TO ?BROTHER ELSE SELF PARENT TO ?CMILD THEN SELF TO BROTHER PARENT , 0 , 0 , 0 , ; (parent/brother/flas/child) : HEADER : FORMAT.HEADER (defines FORMAT relatives 1 executes HEADER) INSTALL L.FIELD IN UNCLE INSTALL C.FIELD IN PARENT INSTALL L.FORM IN BROTHER INSTALL C.FORM IN SELF C.FIELD NEW-SON MEADER O TO C.FIELD ; --; 171 **************** BLOCK *************** 3-JUL-61) (instruction and format defs. GEC : INSTRUCTION (INSTRUCTION <name> WIDTH <width>) O TO C.FIELD FORMAT.HEADER IGNORE GET# DUP ; (instruction width) DUP TO F.LENGTH TO F.POS ; (field length/field position) FORMAT (FORMAT) FURNAI PFIELD.LENGTH TO F.LENGTH PFIELD.START F.LENGTH + TO F.POS FORMAT.HEADER ; SET.FLAGS (<+>-<> sets (field length (field position) (<*>-<> sets flags from C.FIELD up to #) : SELFLAGS TO XFLAG C.FIELD BEGIN TPARENT XFLAG TO OVER TFLAG DUP NOT END DROP XFLAG C.FIELD TO TFLAG # --> 172 **************** format.end and field header GEC 3-JUL-FORMAT.END (END. C.FIELD TARCESTOR DUP TO L.FIELD TO C.FIELD C.FIELD TPARENT IF TFIELD.START ELSE 0 THEN F.POS <> IF 2 ERROR.FUNCT RESTART ELSE TFIELD.LENGTH TO F.LENGTH TFIELD.START TO F.POS THEN : 3-.118 -81) (END.FORMAT) THEN : : FIELD.HEADER INSTALL L.FORM IN UNCLE INSTALL C.FORM IN PARENT INSTALL L.FORM IN UNCLE INSTALL C.FORM IN PARENT INSTALL L.FIELD IN BROTHER INSTALL C.FIELD IN SELF SELF 0<> IF SELF PPARENT ELSE C.FORM THEN DUP TO PARENT NEW.SON HEADER ; ------> 173 **************** error checking for used fields GEC ER.CHECK (check to s 0 TO FLD.FF C.FIELD BEGIN DUP ?FLAG TO TEST.FLAG FLD.FF DROP "PARENT DUP NOT TEST.FLAG OR (if f END DROP IF 4 ERROR.FUNCT RESTART ELSE TEST.FLAG NOT IF 5 ERROR.FUNCT RESTART THEN THEN 0 TO TEST.FLAG ; 3-JUL-01) (check to see if field is permitted) (set TEST.FLAG=FLAG) (flip field.flip.flop) (so to parent) (if flas found or root reached) (field defined twice) (not proper instruction) THEN O TO TEST.FLAG ; ---> ***************** 174 8-JUL-81) : DEFAULT GET# TO.DEF DO.DEFAULT ; -->

w,

when a "(" is encountered and decremented when a ")" is found. This would catch any errors involving too many closing parentheses. A "]" function could be written which would behave in the same manner as the UCI LISP function of the same name. It would use the variable mentioned above to close all open parentheses for a successful evaluation of the expression.

GET# and its related algebraic functions have some interesting features in that there is no hierarchial ordering of functions (i.e., $2 + 3 \cdot 5 = 25$ while $5 \cdot 3 + 1 = 17$), however, expressions enclosed in parentheses will be solved before others .e., $2 + (3 \cdot 5) = 17$). The entire code for this is only a few lines long and is as follows:

GET# 32 WORD NUMBER NOT IF R> R> SWAP >R >R THEN ;

-CCABULARY ALGEBRAIC ALGEBRAIC DEFINITIONS redefine functions

gets number

swap if not a number

	: - GET# - ; : / GET# / :	re-swap return stack
R> R> SWAP >R >R);		swap return stack

FIRTH DEFINITIONS

A typical usage of this function could te:

: {+} GET# + ;

Turrent Function	Command	Parameter Stack	Return Stack	
main	3	3	•	input a 3
{+}	(+)	3	main	call function {+}
GET#	GET#	3	main (+)	call function GET#
	(3	(+) main	swap return stack
main	4	34	{+}	return and input a 4
(+)	(+)	34	(+) main	call (+) again
GET#	5	345	{+} main {+}	input a 5
{ + }	+	39	(+) main	return and add
⊐ain		39	{+}	return to main
))	39	(+) main	call function)
-		39	main {+}	swap return stack
(+)	+	12	main	return and add
nain		-	•	return and print

There are a few general concepts which are used throughout this micro assembler, one of which is the "TO contept" (see Joe Sawicki's paper entitled Optimized Data Structures for Hardware Control). This concept allows the use of variables without the programmer having to deal directly with the address. While this may be thought of as being a bit un-

> 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 whenever 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 essential for the assembler to run, it again makes the code cleaner and easier to use.

Conclusion

The reason why I have chosen to write this micro assembler in FORTH is simplicity. As I mentioned earlier, this "program" 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 completed and the output formatting is implemented, this code will be a micro assembler by itself 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 other symbol tables to allow external references. Through the use of external symbol tables the microcode 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-

arem which would be used, in much the same manner as the micro assembler, and similar to Hardware Description Languages, to describe a simulator for the microcode. 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 code[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.

Acknowledgements

I would like to thank Lawrence Forsley for the time and effort he expended helping to direct and complete this project. I would also like to extend thanks 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-AC08-80DP40124.

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 Engineer

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

176 **************** (end.instr 1 find root 1 brother GEC 11-JUN-81) ND_INSTRUCTION (checks for any undefined fields) BEGIN FORMAT.END C.FIELD ?ANCESTOR NOT END ; : END_INSTRUCTION (finds instruction /
([# self]==[self parent]) : ROOT O SWAP BEGIN DUP ?PARENT ROT DROP DUP NOT END DROP ; : FIND.BROTHER O SWAP (finds brother with flas set) IND RRUTHER O SHAP BEGIN DUF ?BROTHER ROT DROF OVER ?FLAG OVER NOT OK PFLAG NOT IF DROF O THEN ; ([# self]--[self brother]) (flag DK not brother) ([brother OR 0]) --> 177 *************** **************** BLOCK 178 **************** MICFO-2558HULL, LABEL SUILDS 0, 0, (def.flag / val) DOES: NEW.WORD IF DUP @ IF .' Label previously defined' CR RESTART IHEN 1 DVER ! (set flag) 24 LC SWAF ! (set value) ELSE DUF @ IF 24 @ ELSE 7FIELD.START SWAF , 0 THEN ; micro-assembler: forward ref. GEC 17-JUL-81) 179 ***************** 6EC (end of word & origin 11-JUN-81) (ends word in program mode) S C.FIELD ROOT IF DEFAULT! THEN C.FIELD RUDI IF DEFHUL!! IMEN (end of labels) XFRINT.FORMAT , LC DUP , 1+ TD LC PRECISION 0 B0 DUP B. , LOOP CR 0 1 E.FILL 1 TO NEW.WORD ; : ORG GET# TO LC ; 180 *************** printing routine GEC 18-JUN-81) Printing routine GEC U.ZERO DUP 4096T U>≈ IF OT ELSE DUP 256T U>≈ IF 1T ELSE DUP 16T U>≈ IF 2T ELSE DUP 16T U>≈ IF 2T THEN THEN THEN DUP IF DUP OT DO OT 1T U.R LOOP THEN 4T SWAP - U.R ; -->

181 *************** / Printing routines - 2 GEC 16-JUN-81) : #PRINT (<ext.pre.#.addr>-<> print ext.pre.# in binary & hex) DUP FRECISION 2T # + SWAF 2DUP DO I @ B. 2T +LOOP .* : * DO I @ U.ZERO 2T +LOOP # : MEH.INC HEM DUP 24 TO MEN @ ; ----182 *************** Printing routines - 3 1.PASS.PRINT GE C 16-JUN-81) DUF TO MEM @ 1 AND IF .' ERKOR - PROGRAM LENGTH 0 ° CR ELSE 10 BASE ! CR BEGIN MEM @ IF BEGIN · LABEL : ' MEM DUP @ CFA TNAME CR 2+ TO MEM .' SHIFTED: ' MEM DUP @ . CR CR 2+ DUP TO MEM @ NOT THEN MEN 24 TO MEN 'FORMAT: 'NEM DUP @ . CR 24 TO MEN 'FORMAT: 'NEM DUP @ . CR 24 TO MEN MEN \$PRINT CR MEN PRECISION 2* + TO MEN CR CR CR MEN @ 1 = END CR 10T BASE ! THEN ; ÉND ••> *************** c program statement GEC 16-JUN-81) : PROGRAM -BUILDS IGNORE GET# DUP , ?PRECISION TO PRECISION 0 , 1 TO NEW WORD 0 1 E.FILL DOES: DUP @ PRECISION TO PRECISION 4 + 1.PASS.PRINT # --> ################## BLOCK 184 **************** end program 1 Microp commands GEC END.PROGRAM EDROF 1 , ; 17-JUN-81) : MICROP [COMPILE] : # : END.MICROF (COMPILE) # # IMMEDIATE : EQU (EQU <var.name> WITH <expression>) J'E IGNORE GET# SWAP ! # HICRO DEFINITIONS \$S

INDUSTRY NEWS

FOR TH-Based Savvy Lets User Talk to Computer

FORTH, Inc. is working with its parent company, Technology Industries, Inc. of Santa Clara, California, to develop a new software package for the Apple II, using a Z80 processor. With it, the Apple will offer the kind of casual and efficient mancomputer interface that until now, existed only in movies like <u>2001</u> and <u>Star Wars</u>.

The project calls for Savvy--the trade name for Excalibur Technology Corporation's Adaptive Pattern Recognition Processor--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 phrases. (Future versions will understand spoken words and respond to Spanish commands as well as English. Other languages will follow.)
- o Translates these imprecise patterns into precise computer commands.

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 system's full potential and demonstrating 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.

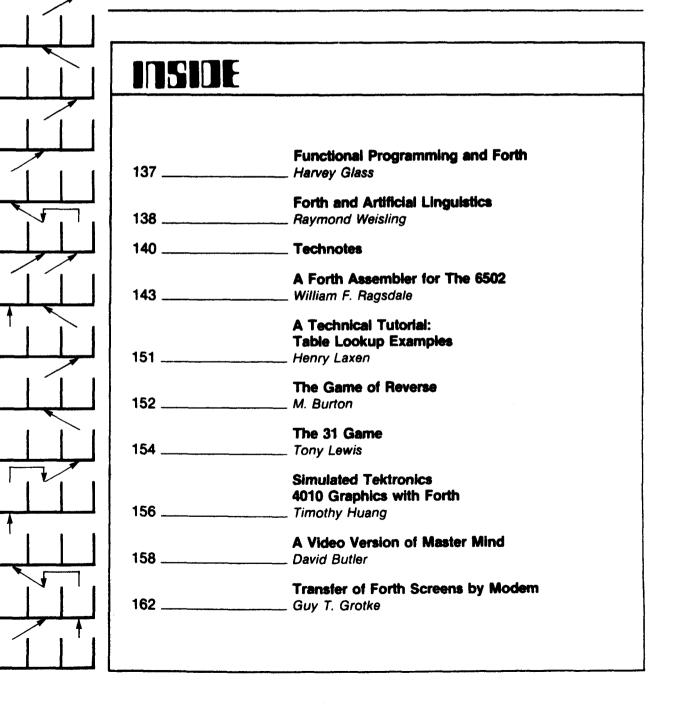
NO ROOM FOR THE ORDER FORM THIS TIME! ORDER - Proceedings 1981 Rochester FORTH Standards Conference. Send check or MO to FIG in US funds on US bank, \$25.00 US, \$35.00 Foreign.





FORTH INTEREST GROUP

P.O. Box 1105 San Carlos, CA 94070 Volume III Number 5 Price: \$2.50



FORTH DIMENSIONS

Published by Forth Interest Group

Volume III No. 5

January/February 1982

Roy C. Martens

C. J. Street

Publisher Editor

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 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 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 Sen 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 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 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

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

I really appreciate all the help and support I have received from our readers. I have not done everything right and some of the best help has been 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 articles designed to help our entry level readers. This, however, will not be done at the expense of our more seasoned FIGGERS who will find an expanded base of challenging articles and applications.

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

C. J. Street

PUBLISHER'S COLUMN

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

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

Roy C. Martens

FOR TH DIMENSIONS III/5

ħq

tions of volunteers serving 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 concepts 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.

Dear Fig:

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

> Bob Royce Box 57 Michiana New Buffalo, MI 49117

Your kit is on the way! Anyone else?

--ed.

Dear Fig:

Glen Haydon's nice article in FORTH DIMENSIONS III/2, page 47 talks about an algorithm he would like to have to determine the Julian day. With the background that FORTH has in astronomy, "m sure there must be several, but this is the nicest I know. It comes from the U. S. Naval Observatory via an article in the Astrophysical Journal Supplement Series, Vol. 41 No. 3 Nov. 1979 pp 391-2.

0	(JULIAN DATE)
1	: JD >R SWAP
2	DUP 9 + 12 / R + 7 * 4 / MINUS
3	OVER 9 - 7 / R + 100 / 1+ 3 + 4 / -
4	SWAP 275 9 */ + 📑
5	+ S-> D 1.721029 D+
6	367 R> M* D+ ;

Example: 3 20 1982 JD D. 2445049 OK

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

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

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

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

FUNCTIONAL PROGRAMMING AND FORTH

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

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

Conventional Languages

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

Functional Programming

This new style of programming has the following characteristics:

- A function (program) is constructed from a set of previously defined functions using a set of <u>functional</u> forms that combine these existing functions to form new ones.

- The most fundamental functional form is called <u>composition</u>. If the composition operator is denoted by o, 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 readers may find it difficult to view them as programming languages." We have a set of predefined functions in a library (dictionary) and may define new functions in terms of these predefined functions.

Functional forms are constructs denoting functions which take functions as parameters. For example, the construct "if-else-then", and the construct "do while" are functional forms. As indicated above, composition is also a functional form.

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

An Example of Functional Programming: The Factorial Function

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

def $\pm \equiv eq0 + \overline{1}; \bullet o [id, \pm subl]$, where the notation $o, \equiv, and []$ denote functional forms. As we have seen, o denotes composition. The notation $[f_1, f_2]$ denotes construction where $[f_1, f_2]$ applied to an argument x yields the sequence $\langle f_1(x), -f_2(x) \rangle$. The notation $p \Rightarrow f_{ij}$ applied to an argument x indicates that the value p(x) is to be examined and if p(x) is true the expression yields f(x) else it yields g(x).

Other definitions used in the above are:

eq0 applied to x yields a value true if x is 0, and yields false otherwise.

 $\overline{1}$ is the literal value 1 and yields the

value 1, regardless of the argument.

is the multiplication operator, and applied to a sequence <x,y> yields x*y.

id is the identity operator. id applied to x yields x.

subl applied to an argument x yields x-1.

Following the logic of the above function we see that "applied to an argument n yields 1 if n is zero. If n is not zero we generate $n^{\pm}(n-1)!$

Clearly then for $\underline{N}0$ this is a definition of the factorial function. In FORTH (if the language were recursive) we would write:

: " DUP 0= 1F 1+ ELSE DUP 1 - " + THEN ;

The syntaxes of the two examples 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 apparent that the style is essentially the same. We have "words" which denote functions which are evaluated following very similar rules.

FORTH as a Language with Characteristics of Functional Programming

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 than that of conventional languages--that provides a power and flexibility that has sparked the enthusiaam of so many of us.

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 particularly 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 result of this paper, languages which have attributes similar to FORTH begin to appear in academic circles.

- The author has recently implemented such an operator in FORTH.
- ** The way that literals are handled can be viewed as merely a question of implementation and efficiency.

References

1. J. Backus, "Can Programming be Liberated from the Von Neumann Style?" CACM, Vol. 21, No. 8, August 1978, p. 613.

FORTH AND ARTIFICIAL LINGUISTICS

Raymond Weisling Surakarta, Jawa Tengah Republik Indonesia

There 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 language, 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 extensibility and hierarchal function of operators.

The point I wish to address here is the syntactical limitations of the language we are building, an artificial language based in part on a human language (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 syntax-sensitivity of word forms, especially nouns and verbs, which in English are commonly spelled and pronounced exactly the same. We rely on the structure (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 , LIMIT , 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 verbs) 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 Armchair 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 dif-Those here who learn ferentiators). 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.

1

4

á

3

4

745

s

53

30

st:

and

FORTH STANDARDS CORNER Robert L. Smith

More Words on WORD

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

: WORD WORD HERE ;

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

- (1) Implementations which place the string at HERE could be considered non-standard, and the problem goes away.
- (2) A clarification could be added to the Standard indicating either that the string will always be at HERE, or that it may be at HERE.
- (3) The problem could be forced upon users by requiring that the characters from WORD be stored in a user-defined area prior to their movement to the final destination.

_et Me Number the Ways

In many areas the 79-Standard defines mits and formats in painful detail. There s an important area in which very little is said, namely the format for single and touble precision numbers in the input stream. In the section "interpreter, text" t is clear that "numbers" are allowed in the input text stream and may either be compiled or placed on the parameter stack. A definition of the format of a number should include at a minimum the 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 ne spirit of the rest of the Standard, I -ould like to propose a few definitions which should be fairly easy to implement and which appear to be compatible with most current implementations (including fig-FORTH). First, we define a digit:

digit

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

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

number

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

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

Under the Spreading FIG-TREE

As many of you are aware, there is a Computer Conference Tree (now nicknamed the FIG-TREE) which contains items of interest to the FORTH community. I would like to encourage all persons interested in the 79-Standard to read and contribute to the branch of the FIG-TREE called 79-STANDARD. All you need is a terminal (110 or 300 baud), a modem, and a telephone. The number is (415) 538-3580. See back issues of FORTH DIMENSIONS for further information, or just call up and send a few carriage returns until the system responds.

CORRECTIONS

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

REFERENCES

- Forsley, Lawrence P. <u>The Beta</u> <u>Laser Control System</u>. A talk given at the Laboratory for Laser Energetics on March 9, 1977 and on July 16, 1977 at the Wilson Synchrotron, Cornell University.
- Forsley, Lawrence P. "Forth Multitasking in URTH". <u>The Best of the</u> <u>Computer Faires</u> Volume <u>IV.</u> 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 <u>QE-17</u> No. <u>9</u>. New York, New York: IEEE, September, 1981.

- 4. <u>Towards More Usable</u> <u>Systems: The LSRAD Report.</u> (Large Systems Requirements for Application Development). Chicago: Share, Inc., 1979.
- 5. _____ IEEE Standard 583-1975. New York: IEEE, 1975.
- <u>1977 Laboratory for Laser</u> Energetics <u>Annual Report</u>. Rochester, NY: <u>Laboratory for</u> Laser Energetics, 1978.
- Moore, Charles. "Forth: A New Way to Program Minicoimputers" Journal of Astronomy and Astrophysics Supplement <u>15</u>. New York: AAAS, September, 1974.
- Moore, Charles. "Forth, The Past Ten Years, and the Next Two Weeks". Forth Dimensions. Vol. <u>16</u> San Carlos, CA: Forth Interest Group, 1979.
- Rather, Elizabeth and Charles Moore. "The FORTH Approach to Operating Systems". <u>ACM 76 Proceedings</u>. New York: ACM, October, 1976.
- Ritchie, D. M. and K. Thompson. "The UNIX Time-Sharing System". The Bell System Technical Journal. Vol. <u>57</u> No. <u>6</u> Part 2. New Providence, NJ: A.T. and T., July-August, 1978.
- Ritchie, D. M., et al. "The C Programming Language". The Bell System Technical Journal. Vol <u>57</u> No. <u>6</u> 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 Dr. Paul Bartholdi² as an alternative to constants and variables.

EDITOR'S NOTE:

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

TECHNOTES, BUGS AND FIXES

I have three questions about FORTH:

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

SCREEN: 95

(HUNT FOR CONTROL CHARACTERS) : HUNT (SCREEN # ----) BLOCK 1024 0 DO DUP Ca DUP 32 < IF CR ."+" 64 + EMIT @ " DUP U. ELSE DROP ENDIF 1+ LOOP DROP; : FIXSCREEN (SCREEN # ----) BLOCK 1024 0 DO DUP Ca 32 4 IF DUP 32 SWAP C! ENDIF 1+ LOOP DROP ; (ACTUALLY HUNT AND FIXSCREEN ARE QUITE SIMILAR, HUNT JUST SHOWS UP ANY GUILTY CHARACTERS AND FIXSCREEN REPLACES THEM)

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

THAT MYSTERIOUS fig-FORTH AMNESIA

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

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

In a dictionary search, the address in the link-field is followed to the beginning of the name-field of the previous word. If it is not a match to the key you are looking for, we scan forward in memory until the most significant bit tells us we have found the link-field to the next word. When a dictionary link is "broken" by 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 64K, it may address RAM where there are no boards). Since it is not likely to find a match at this address, it scans forward for the most significant bit that marks the end of the "name". The odds are that it will eventually find one, mistake the next two bytes for a link, and follow another wild address somewhere else.

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

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

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

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

TRANSIENT DEFINITIONS

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

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

Philip Wasson

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 <u>function</u> 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.

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 Therefore, "unbounded" free space. whether WORD handles tokens of length 128, 256 or even 1024 bytes is innocently discussed with the idea that the only issue involved is the length descriptor preceding the string!

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

:: :

19 11 11

e

e

t ::

ы

:5

rs.

Ŧ.

ise

re-

255

me

rhe

ħe-

15

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

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

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

If you discard the notion that a more or less unbounded "free space" exists somewhere in memory, the approach to WORD's definition takes on a new facet. At Microsystems, we have developed several large applications using FORTH, which resulted in target compiled code in the range of 32K to 48K bytes, exclusive of the dictionary headers and the FORTH operating 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 proFORTHTM, handles this problem by means of multiple dictionaries and ROM/RAM segment control with selective symbol purging. Names are classified as to their needed lifetimes during compilation. When the names are no longer needed, they are purged and their memory space is reclaimed. This allows much of the memory devoted to dictionary headers to be reused many times during compilation, thereby enabling very large applications to be compiled.

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

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

I (WORD) (c -- a n) BLK • 7DUP IF BLOCK FLSE TIB • THEN >IN • + SWAP ENCLOSE >IN +! OVER - -ROT + SWAP ;

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

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

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

WORD, (c --)
(WORD) HERE OVER 1+ ALLOT SWAP OVER C1
COUNT CNOVE;

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 C: PAD
COUNT CMOVE ;

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

: WORD (c -- a) (WORD) WDSZ MIN WBFR C1 WBFR COUNT 1* CMOVE 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 (or 16 bit word if you prefer). In other words, the user ought to be able to do essentially anything that he may desire, but, the implementor need not provide any special, temporary buffers or arbitrary size just to satisfy the Standard.

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

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

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

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

CORRECTION TO FEDIT

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

- 1. SCR 64 Line 10: compile should be COMPILE
- 2. SCR 65 Line 23: 1+/MOD should be 1+ 16 /MOD
- 3. SCR 67 Line 48: B/BUD should be B/BUF
- 4. SCR 67 Line 49: :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 FIG 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 Fig-FORTH line editor definition, R. I used the pseudonym because FEDIT was the first program I wrote in FORTH and I really wasn't familiar enough with 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 Calendar Court La Grange, IL 60525

A HELPFUL UTILITY

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

DECIMAL ; CVD BASE @ SWAP OVER /MOD ROT /MOD 10 • + 10 • +

;

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 system's in hex. If I do the following:

: LIST CVD LIST ;

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

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

> Gregg Williams BYTE Publications PO Box 372 Hancock, NH 03449

CALL FOR PAPERS

1982 Rochester FORTH Conference

Data Bases and Process Control

May 17 through May 21, 1982

University of Rochester Rochester, New York

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

There is a call for papers on the following topics:

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

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

For more information, please contact the conference chairman at:

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

A FORTH ASSEMBLER FOR THE 6502 by William F. Ragsdale

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

Imagine having an assembler in 1300 bytes of object code with:

- User macros (like IF, UNTIL,) definable at any time.
- 2. Literal values expressed in any numeric base, alterable at any time.
- 3. Expressions using any resident computation capability.
- 4. Nested control structures without labels, with error control.
- 5. Assembler source itself in a portable high level language.

OVERVIEW

£

Forth is provided with a machine language assembler to create execution procedures that would be time inefficient, if written as colon-definitions. It is intended that "code" be written similarly to high level, for clarity of expression. Functions may be written first in high-level, tested, and then re-coded into assembly, with a minimum of restructuring.

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 sequence is the usual 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 stream (not in the compile mode). These assembler words specify operands, address modes, and op-codes. At the conclusion of the CODE definition a final error check verifies correct completion by "unsmudging" the definition's name, to make it available for dictionary searches.

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 instant is called 'assembling' or 'assembly-time'. This function may involve op-code generation, address 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.

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.

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:

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

NEXT

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

RETURN OF CONTROL

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

SECURITY

Numerous tests are made within the assembler for user errors:

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

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

i.e., 0= of ASSEMBLER when you want 0= of FORTH (Editor's note: the listing assumes that the figFORTH error messages are already available in the system, as follows:

?CSP issues the error message "DEFI-NITION 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 arguments do not match.

3 ERROR prints the error message "HAS INCORRECT ADDRESS MODE".)

SUMMARY

The object code of our example is:

305 983 4D 4F CE 305D 4D 30	CODE MON link field
305F 61 30 3061 00	code field
3062 4C 42 02	BRK JMP NEXT

OP-CODES, revisited

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

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

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

The multi-mode op-codes are:

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

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

Sym	bol Mode	Operand
. A	accumulator	none
#	immediate	8 bits only
,×	indexed X	z-page or absolute
,Y	indexed Y	z-page or absolute
X)	indexed indirect X	z-page only
)Y	indirect indexed Y	z-page only
)	indirect	absolute only
none	memory	z-page or absolute

EXAMPLES

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

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

(.A distinguishes from hex number 0A)

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

6502 CONVENTIONS

Stack Addressing

The data stack is located in z-page, usually addressed by "Z-PAGE,X". The stack starts near \$009E and grows downward. The X index register is the data stack pointer. Thus, incrementing X by two removes a data stack value; decrementing X twice makes room for one new data stack value.

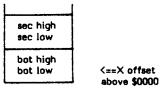
Sixteen bit values are placed on the stack according to the 6502 convention; the low byte is at low memory, with the high byte following. This allows "indexed, indirect X" directly off a stack value.

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

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

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

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



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

BOT LDA,	LDA (0,X)
BOT 1+ ORA,	ORA (1,X)
SEC ORA,	ORA (2, X)
SEC 1+ ORA,	ORA (3,X)

To obtain the 14-th byte on the stack: BOT 13 + LDA,

RETURN STACK

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

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

Return stack values may be obtained by: PLA, 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 XSAVE
- 2) execute TSX, to bring the S register to X.
- 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.

CODE IS-IT (zero?)

XSAVE STX, TSX, (setup for return stack) RP) 2+ LDA, RP) 3 + ORA, (or 2nd item's two bytes

- together)
- 0= IF, INY, THEN, (if zero, bump Y to one)
 - TYA, PHA, XSAVE LDX, (save low byte, rstore data stack) PUSH JMP, END-CODE (push boolean)

	Return Stack	
PR) =\$0101,X>	hi byte lo byte	second item
	hi byte lo byte	bott om item
S>	fr ee byte	

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.
- address of the pointer to the code field of the dictionary definition just interpreted by NEXT. W-1 contains \$6C, the op-code for indirect jump. Therefore, jumping to W-1 will indirectly jump via W to the machine code for the definition.
 - UP User Pointer containing address of the base of the user area.
 - a utility area in z-page from N-1 thru N+7.

CPU Registers

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

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

XSAVE

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

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

N Area

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

The assembler word N returns the base address (usually \$00D1). The N Area spans 9 bytes, from N-1 thru N+7. Conventionally, N-1 holds one byte and N, N+2, N+4, N+6 are pairs which may hold 16-bit values. See SETUP for help on moving values to the N Area.

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

CODE DEMO HEX 6 # LDA, N 1 - STA, (setup a counter)

BEGIN, 8001 BIT. (tickle a port)

> N 1 - DEC. (decrement the counter)

0= UNTIL, NEXT JMP, END-CODE (loop till negative)

SETUP

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

			,
stad	ck before	N after	stack after
	H high		н
	G low	bo	ot> G
	F ⁻	F	
	E D	ε_	
	D	ס	
sec>	С	c_	
	B	в	
bot>	Α	N> A	

3 # LDA, SETUP JSR.

CONTROL FLOW

Forth discards the usual convention of assembler labels. Instead, two replace-ments 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,	UNTIL,	IF,	ELSE,
THEN,			

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

One major difference occurs! 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,).

Examples are:

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

The conditional specifiers for 6502 are:

CS	test carry set	C=1 in
	processor	
	status	
0K	byte less than zero	N=1
0=	equal to zero	Z=1
CS NOT	test carry clear	C=0
0 KNOT	test positive	N=0
0= NOT	test not equal zero	Z=0

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

ASSEMBLER DEFINITIONS HEX 50 CONSTANT VS (test overflow set)

CONDITIONAL LOOPING

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

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

(N reaches zero)

First, the byte at address N is loaded with the value 6. The beginning of the loop is marked (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.

CONDITIONAL EXECUTION

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

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

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

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

PORT LDA, 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:

PORTLDA, OKIF, N DEC,	(fetch one byte) (if D7=1, decrement
ELSE, NINC,	N) (if D7=0, increment N)
THEN,	(continue ahead)

CONDITIONAL NESTING

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

BEGIN, < code always executed>

CS IF, <code if carry set>

ELSE, <code if carry clear> THEN, 0= NOT UNTIL, (loop till condition

flag is non-zero) <code that continues onward>

Next is an error that the assembler security will reveal.

BEGIN, PORT LDA, 0= IF, BOT INC, 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 "conditionals not paired".

RETURN OF CONTROL, revisited

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

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

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

CODE INVERT (a memory byte) HEX
BOT X) LDA, (fetch byte addressed
by stack)
FF # EOR, (complement 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 placing it on the stack by:

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

A simpler version could use PUSH:

CODE ONE

FND-CODE

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

The convention for PUSH and PUT is:

- 1. push the low byte onto the machine stack.
- 2. leave the high byte in the accumulator.
- 3. jump to PUSH or PUT.

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

FOOLING SECURITY

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

BEGIN, UNTIL,		addr8 1	cc	 addr8 1
			cc	addrl 2
ELSE,	==>	addrl 2		 addrE 2
THEN,	==>	addrl 2		
	or	addrE 2		

The address values indicate the machine location of the corresponding 'B'EGIN, 17F, or 'E'LSE,. cc represents the condition code to select the processor status bit referenced. The digit 1 or 2 is tested for conditional pairing.

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

To generate the equivalent of the high level:

BEGIN (a) WHILE (b) REPEAT

we write in assembly:

BEGIN, DROP (the check digit 1, leaving addrB)
<a> CS IF, (leaves addrI and digit 2)
(b) ROT (bring addrB to bottom) JMP, (to addrB of BEGIN,) ENDIF, (complete false for- ward branch 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 and addresses are correctly available.

ASSEMBLER GLOSSARY

- Specify 'immediate' addressing mode for the next op-code generated.
-)Y Specify 'indirect indexed Y' addressing mode for the next opcode generated.

- ,X Specify 'indexed X' addressing mode for the next op-code generated.
- ,Y Specify 'indexed Y' addressing mode for the next op-code generated.
- A Specify accumulator addressing BOT mode for the next op-code generated.
- 0< --- cc (assembling) Specify that the immediately following conditional will branch based on the processor status bit being negative (Z=1), i.e., less than zero. The flag cc is left at assembly time; there is no runtime effect on the stack.
- 0= --- cc (assembling) Specify that the immediately following conditional will branch based on the processor status bit being equal to zero (Z=1). The flag cc is left at assembly time; there is no run-time effect on the stack.
- ;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 AS-SEMBLER, 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 <namex> . That is, when
<namex> is executed, the address
interpreter jumps to the code following ;CODE in <name>.

ASSEMBLER in FORTH Make ASSEMBLER the CON-TEXT vocabulary. It will be searched first when the input stream in interpreted.

BEGIN, --- addr 1 (assembling) --- (run-time) Occurs in a CODE definition in the form:

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

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

--- n (assembling) Used during code assembly in the form:

BOT LDA, or BOT 1+ X) STA,

Addresses 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 offset into the data stack. Must be followed by a multi-mode op-code mnemonic.

CODE A defining word used in the form:

CODE <name> END-CODE

to create a 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 n ---- (compiling assembler) An assembler defining word used to crate assembler mnemonics that have only one addressing mode:

EA CPU NOP,

CS

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 (assembling) 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, --- (run-time) addr1 2 --- addr2 2 (assembling)

Occurs within a code definition in the form:

cc IF, <true part> ELSE, <false part> THEN, At run-time, if the condition code specified by cc is false, execution will skip to the machine code following ELSE, At assembly time ELSE, assembles a forward jump to just after THEN, and resolves a pending forward branch from IF. The values 2 are used for error checking of conditional pairing.

END-CODE

IF,

IP

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

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

time)

Occurs within a code definition in the form:

cc IF, <true part> ELSE,

false part THEN, At run time, IF, branches based on the condition code cc, (OK or D=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.
 - --- addr (assembling) Used in a code definition in the form:

IP STA, or IP)Y LDA,

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

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

IP STA, or IP)Y LDA,

loads the third byte ahead in the colon-definition being interpreted.

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

1C6E 60 M/CU ADC,

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

- MEM Used within the assembler to set MODE to the default value for direct memory addressing, z-page.
- MODE --- addr A variable used within the assembler, which holds a flag indicating the addressing mode of the op-code being generated.
- N --- addr (assembling) Used in a code definition in the form:

N 1 - STA, or N 2+)Y ADC,

A constant which leaves the address of a 9 byte workspace in zpage. Within a single code definition, free use may be made over the range N-1 thru N+7. See SETUP.

- NEXT --- addr (assembling) A constant which leaves the machine address of the Forth address interpreter. All code definitions must return execution to NEXT, or code that returns to NEXT (i.e., PUSH, PUT, POP, POPTWO).
- NOT 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 --- addr (assembling) n --- (run-time)

A constant which leaves (during assembly) the machine address of the return point which, at runtime, will pop a 16-bit value from the data stack and continue interpretation. POPTWO

PUT

RP)

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

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

--- (assembly-time) Used in a code definition in the form:

RP) LDA, or RP) 3+ STA,

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.

- SEC --- n (assembling) Identical to BOT, except that n=2. Addresses the low byte of the second 16-bit data stack value (third byte on the data stack).
- THEN, --- (run-time) addr 2 --- (assembly-time) Occurs in a code definition in the form:

cc 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,. UNTIL,

UP

w

X)

, --- (run-time) addrl cc --- (assembling) Occurs in a CODE definition in the form:

BEGIN, . . . cc UNTIL,

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

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

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

UP LDA, or UP)Y STA,

A constant leaving at assembly time the address of the pointer to the base of the user area. i.e.,

HEX 12 # LDY, UP)Y LDA,

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 any byte in the definition's parameter field. i.e.,

2#LDY, W)YLDA,

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.

```
FORTH Assembler for 6502 by W. F. Ragadale
                                                                    July 1, 1980
SCR # 81
                                                                              WFR-79JUN03 )
  0 ( FORTH-65 ASSEMBLER
  I HEX
  2 VOCABULARY ASSEMBLER IMMEDIATE
                                                          ASSEMBLER DEFINITIONS
  3
  4 (REGISTER ASSIGNMENT SPECIFIC TO IMPLEMENTATION )
5 EQ CONSTANT XSAVE DC CONSTANT W DE CONSTANT UP
  5 EO CONSTANT XSAVE DC CONSTANT W
6 D9 CONSTANT IP D1 CONSTANT N
  8 ( NUCLEUS LOCATIONS ARE IMPLEMENTATION SPECIFIC )
9 * (DO) OF + CONSTANT POP
6 ( NUCLEUS LUCATIONS ARE IMPLEMENTATION

9 ° (DO) OE + CONSTANT POP

10 ° (DO) OC + CONSTANT POPTWO

11 ° LIT 13 + CONSTANT PUT

12 ° LIT 11 + CONSTANT PUSH

13 ° LIT 18 + CONSTANT NEXT

14 ° EXECUTE NFA 11 - CONSTANT SETUP

15
 15
SCR # 82

      0 (
      ASSEMBLER, CONT.

      1 0
      VARIABLE INDEX
      -2 ALLOT

      2 0909 , 1505 , 0115 , 8011 , 8009 , 1DOD , 8019 , 8080 ,
      3 0080 , 1404 , 8014 , 8080 , 8080 , 1COC , 801C , 2C80 ,

                                                                              WFR-780CT03 )
  5 2 VARIABLE MODE
                                                                  : MEN 2 MODE 1 ;
                                                                : X) 5 HODE 1 ;
                               0;
 10 : BOT
                       , X
                                            ( ADDRESS THE BOTTOM OF THE STACK *)
                      , X
                       ,X 2;
,X 101;
 11 : SEC
                                                ( ADDRESS SECOND ITEM ON STACK *)
                                              ( ADDRESS BOTTON OF RETURN STACK *)
 12 : RP)
 13
 14
 15
SCR # 83
 O ( UPMODE, CPU
                                                                              WFR-780CT23 )
  3
  2 : UPMODE IF MODE @ 8 AND 0- IF 8 MODE +1 TEEN THEN

3 1 MODE @ OF AND ~DUP IF 0 DO DUP + LOOP THEN

4 OVER 1+ @ AND 0= ;
  5
  6 : CPU <BUILDS C, DOES> C@
7 00 CPU BRK, 18 CPU CLC,
                                                   C, MEH ;
D8 CPU CLD, S8 CPU CLI,
            B8 CPU CLV,
                                 CA CPU DEX,
  8
                                                     88 CPU DEY,
                                                                       ES CPU INX,
  9
            C8 CPU INY,
                                 EA CPU NOP,
                                                     48 CPU PHA, OS CPU PHP,
 10
            68 CPU PLA,
                                28 CPU PLP,
                                                    40 CPU RTI, 60 CPU RTS,
            38 CPU SEC,
                                                    78 CPU SEI, AA CPU TAX,
8A CPU TXA, 9A CPU TXS,
                                 F8 CPU SED,
 11
 12
            A8 CPU TAY,
                                BA CPU TSX,
            98 CPU TYA,
 13
 14
```

FORTH DIMENSIONS III/5

15

SCR # 84 O (M/CPU, MULTI-MODE OP-CODES DOFS) WFR-79MAR26) VU <BUILDS C, , DOES> DUP 1+ @ 80 AND IF 10 MODE +1 THEN OVER 1 : M/CPU 2 FF00 AND UPHODE UPHODE IF MEM CR LATEST ID. 3 ERROR THEN C@ MODE C@ INDEX + C@ + C, MODE C@ 7 AND IF MODE C@ OF AND 7 < IF C, ELSE, THEN THEN MEM; 3 4 5 6 7 1C6E 60 H/CPU ADC, 1C6E 20 M/CPU AND, 1C6E CO M/CPU CMP, 8 1C6E 40 M/CPU BOR, 1C6E AO M/CPU LDA, 1C6E 00 M/CPU ORA, 9 ODOD 01 M/CPU ASL, 1C6C 80 M/CPU STA, 1C6E EO M/CPU SBC, 10 OCOC E1 M/CPU INC, 11 OCOC CI M/CPU DEC, ODOD 41 M/CPU LSR, ODOD 21 M/CPU ROL, ODOD 61 M/CPU ROR, 0414 81 M/CPU STX, 12 0486 CO M/CPU CPY, 1496 A2 M/CPU LDX, 0486 EO M/CPU CPX, 13 OCSE AO M/CPU LDY, 048C 80 M/CPU STY, 0480 14 M/CPU JSR, 14 0484 20 M/CPU BIT. 15 8480 40 M/CPU JMP, SCR # 85 O (ASSEMBLER CONDITIONALS WFR-79MAR26) 1 : BEGIN, HERE 1 ; 2 : UNTIL, ?EXEC >R 1 ?PAIRS R> C, HERE 1+ - C, ; IMMEDIATE TMMEDIATE C, HERE O C, 2 ; IMMEDIATE ?EXEC 2 ?PAIRS HERE OVER C@ IF SWAP ! ELSE OVER 1+ - SWAP C1 THEN ; IMMEDIATE 2 ?PAIRS HERE 1+ 1 JMP, 4 : THEN, 5 6 : ELSE, SWAP HERE OVER 1+ - SWAP CI 2 CI 2 ; IMMEDIATE (REVERSE ASSEMBLY TEST) 7 8 : NOT 20 + ; (REVERSE ASSEMBLY TEST) 9 90 CONSTANT CS (ASSEMBLE TEST FOR CARRY SET) 10 D0 CONSTANT 0- (ASSEMBLE TEST FOR EQUAL ZERO) 11 10 CONSTANT 0< (ASSEMBLE TEST FOR LESS THAN ZERO) 12 90 CONSTANT >= (ASSEMBLE TEST FOR GREATER OR EQUAL ZERO) 12 00 CONSTANT >= (ASSEMBLE TEST FOR GREATER SUB, OR CMP,) 13 (>= IS ONLY CORRECT AFTER SUB, OR CMP,) 14 15 SCR # 86 O (USE OF ASSEMBLER WFR-79APR28) 1 : END-CODE (END OF CODE DEFINITION *) CURRENT @ CONTEXT i ?EXEC ?CSP SMUDGE ; IMMEDIATE 2 3 4 FORTH DEFINITIONS DECIMAL 5 : CODE (CREATE WORD AT ASSEMBLY CODE LEVEL *) **?EXEC CREATE [COMPILE] ASSEMBLER** ASSEMBLER MEM ICSP ; IMMEDIA 6 7 INMEDIATE 8 9 (LOCK ASSEMBLER INTO SYSTEM) 10 ° ASSEMBLER CFA ° ;CODE 8 + ! (OVER-WRITE SMUDGE) 11 LATEST 12 +ORIGIN ! (TOP NFA) 12 HERE 28 +ORIGIN ! (FENCE) 13 HERE 30 +ORIGIN ! (DP) 14 ° ASSEMBLER 6 + 32 +ORIGIN ! (VOC-LINK) 15 BERB FENCE I

2001-012

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.

APPLICATIONS

A TECHNICAL TUTORIAL:

TABLE LOOKUP EXAMPLES

Henry Laxen

Laxen and Harris. Inc.

One of the problems with FORTH, as with every rich language, is that given an

This article is written to illustrate 4 different ways of implementing a simple Table Lookup operation. The goal is the following: we want to create a FORTH word, named DAYS/MONTH which behaves as follows: Given an index on the stack which is the month number, such as 1 for January and 12 for December, we want to return the number of days in that month, in a normal year. Thus if we execute 6 DAYS/MONTH it should return 30, which is the number of days in the month June. I will use the Starting FORTH dialect in this paper, not fig-FORTH, so if you try to type in the examples, they probably won't work unless you are running a system that behaves as described in Starting FORTH (or the 79-Standard).

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

VARIABLE 'DAYS/MONTH 22 ALLOT

31 'DAYS/MONTH			:	
28 'DAYS/MONTH	2	+	!	
31 'DAYS/MONTH	4	+	:	
30 DAYS/MONTH	6	+	1	
31 'DAYS/MONTH	8	+	:	
30 DAYS/MONTH	10	+	:	
31 'DAYS/MONTH	12	+	:	
31 'DAYS/MONTH	14	+	:	
30 DAYS/MONTH	16	+	1	
31 'DAYS/MONTH	18	+	:	
30 'DAYS/MONTH	20	+	:	
31 'DAYS/MONTH	22	+	1	
: DAYS/MONTH (I	NDE	x	- VALUE)
1- 2* 'DAYS/M	ONT	H *	a ;	·

There is nothing significant about the ' (apostrophe), I only prefaced the VARI-ABLE name with it because I want to use the word DAYS/MONTH later. Now, what happened is that VARIABLE allocated 2 bytes in the dictionary for the value of DAYS/MONTH. The 22 ALLOT then allocated another 22 bytes, for a total of 24 bytes, or 2*12 cells. We next proceeded to initialize the values that were allocated by explicitly calculating the offsets and storing in the appropriate location. Finally, we defined DAYS/MONTH as a colon definition which performs arithmetic on the index, adds it to the start of the table, and fetches the result.

Now, let's look at another way of doing

this that requires less typing and is also more general. We will first define a word called TABLE which will aid us in the creation of tables like the one above. What we will do is first place the initial values of the TABLE on the stack, together with the number of the initial values. Then, we will define TABLE to copy these into the dictionary. Here is how it works:

:	TABLE	(Nn Nn-1 M	vl n)	
	ODO.	LOOP;		

- CREATE 'DAYS/MONTH 31 30 31 30 31 31 30 31 30 31 28 31 12 TABLE
- : DAYS/MONTH (INDEX --- VALUE) 1- 2* 'DAYS/MONTH + @;

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

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

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

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

Now we are getting somewhere." If we simply use the FORTH word, (comma) to add the value to the dictionary, see how simple and readable it becomes. The values are just typed in and separated by commas." Is it possible to improve even on this? 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 (---) DOES> (INDEX --- VALUE) SWAP 1- 2* + @;

 TABLE DAYS/MONTH

 31, 28, 31, 30, 31, 30,

 31, 31, 30, 31, 30, 31,

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 new words. Part of the definition of TABLE was specifying the run-time behavior of the word being defined. This is the code following the DOES. 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 diversity available in FORTH. There is no obvious right or wrong, and the simplest and usually most general solution to a given problem must be discovered, usually by trial and error. FORTH's biggest virtue, in my opinion, is that it makes the trial and error process extremely efficient, and therefore, allows people to experiment and discover the best solution for themselves.

HELP WANTED

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

Contact: Patricia Jones

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

fig-FORTH NOVA GROUP

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

Pub. Comment: Hope to see a new, clean listing. How about some other specific groups!

THE GAME OF REVERSE

M. Burton

REVERSE is a number game written in SCR • 228 O (The Game of Reverse (SEED, NOVES, RND, DIN, Y/N) 101201-MPB) FORTH, primarily as an exercise in array manipulation. The object of REVERSE is 1 to arrange a list of numbers (1 through 9) VARIABLE SEED VARIABLE NOVES (Seed for random number generator) 20 in ascending numerical order from left to (Number of reverses so far) 3 0 right. Moves are made by reversing a sub-." Please depress any key:" (Fertilize the seed) 4 CR set of the list (from the left). For KEY SEED I 5 example, if the current list is (Random number generator range -- rnd0) • 3 + 32767 AND DUP SEED ! 32767 •/ ; 7 : RND 234516789 SEED 0 259 ٠ 8 ٩ (Reserve an integer word array n --) and four numbers are reversed, the list 10 : DIM 2 • ALLOT will be CBUILDS 1+ 11 <2300 12 : 543216789 13 (Get a Y or N response -- flag) : Y/N 14 PAD BO EXPECT PAD CO CR CR 95 AND then if five numbers are reversed, the 89 = 2 15 dame is won. 123456789 SCR # 229 0 (The Game of Reverse [Game instructions] 101281-MPB) To leave a game that is in progress, 2 : INSTRUCT CR CR simply reverse zero numbers. 10 SPACES ." The Game of REVERSE" ." Would you like instructions? " Y/N CR CR з ." The object of the game is to arrange a random list" **REVERSE Glosserv** IF 4 CR ." of nine numbers into ascending numerical order in" 5 CR ." of nine numbers into ascending numerical order in CR ." as few moves as possible by reversing a subset of" CR ." the list. For example, given the random list," CR CR ." 5 2 4 8 7 3 9 1 6 " CR CR ." reversing a subset of 4 would yield the list," CR CR ." 8 4 2 5 7 3 9 1 6 " CR SEED 6 The number seed for the pseudorandom 7 number generator. SEED is initialized 8 as the REVERSE words are compiled, 9 by hitting any key on the console. 10 ." To quit the game, simply reverse 0." CR CR CR 11 MOVES 12 THEN Keeps track of the number of moves 13 made in a REVERSE game. If more 14 --> than fifteen moves are made to win, 15 you haven't got the hang of the game. RND range -- random.number SCR 0 230 The pseudorandom number generator, 0 (The Game of Reverse [ARRAY operations] 101281-NPB) courtesy of FORTH DIMENSIONS. RND generates random.number in the DIM ARRAY (Reserve a ten word array) 29 range 0 through range-1. RND is used 3 to scramble the number list. (Fetch an array word index -- array.value) 4 : 30 ARRAY + 0 : 5 DIM n --6 A defining word used in the form array.value\index --) 7 : A1 (Store an array word n DIM xxxx 2 • ARRAY + 1 ; Produces an n+1 length word array 9 named xxxx, with elements 0 through (Initialize ARRAY --) 10 : AINIT n. For the REVERSE application, 10 1 DO I DUP AI LOOP ; 11 element 0 is not used. 12 (Print ARRAY -- } 13 : A -- flag The list is now Y/N 14 C DO I AO 3 .R LOOP : --> Solicits an input string from the con-6 SPACES 10 1 15 CR sole, then checks the first character of the string for an uppercase or lowerfig-FORTH Version 1.15 M. Burton

case 'Y'. If a 'Y' is present, the flag SCR 0 231 0 (The Game of Reverse (ARRAY operations, cont.) 100781-MPB) returned is true. For any other character, the flag is false. 1 : ASCRAMBLE (Mix up the array values -- 1 2 INSTRUCT I RND 1+ (Calculate K) 3 1 9 DO Prints the name of the game and then Aŧ Get ARRAY[I] value) T 4 (asks if instructions are required. If 5 OVER A. (Get ARRAY[K] value) (Store ARRAY(K) in ARRAY(I)) yes, instructions are displayed. 6 T At. SAN A! -1 (Store ARRAY(I) in ARRAY(K)) 7 ARRAY +LOOP : 9 A ten word array that contains the number list that REVERSE works on. : GETIN (Get amount to reverse 10 -- n) ." Reverse how many? " CR Element zero of the list is not used. BEGIN 11 PAD SO EXPECT PAD 9 48 DUP O< OVER 9 > OR DUP 12 13 index -- array.value 14 IF CR ." Only 0 through 9 is allowed. " THEN Fatches the index array.value of 15 UNTIL ARRAY and leaves it on the data CR --> : stack. SCR # 232 A! array.value index --0 (The Game of Reverse | [ARRAY operations, cent.] 100781-MPB) Stores array.value into the index element of ARRAY. 1 (Reverse a subset of ARRAY n -- 1 : AREVERSE 2 3 DUP 2 1+ 1 (Loop limits are 1 to (n/2)+1) AINIT DO DUP 4 1+ (Calculate index n-I+1) Initializes ARRAY with the numbers 1 I (Get ARRAY(n-I+1)) 5 DUP 20 SHAP through nine in game winning order. (Get ARRAY[1]) 6 T 20 (Store ARRAY[I] in ARRAY[n-I+1]) 7 SHAP At 8 At (Store ARRAY(n-I+1) in ARRAY(I)) Displays ARRAY in an understandable Ι 9 LOOP DROP format. : 10 : ACHECK -- flag) ASCRAMBLE 11 (Check for ascending seq. Using RND, scrambles the numbers in 1 10 1 DO 12 13 T DUP 20 = AND ARRAY for a new REVERSE game. L007 ; 14 15 --> GETIN -- n Solicits the number of elements of the list to reverse. If any character other SCR # 233 than 0 through 9 is entered, GETIN (The Game of Reverse (REVERSE definition) 101281-MPB) prints "Only 0 through 9 is allowed.", 0 and solicits another number. 1 : REVERSE 2 (Play the game) INSTRUCT AINIT AREVERSE 3 n ---Reverses the nth length subset of 4 BEGIN 5 ASCRAMBLE O HOVES ! ARRAY, starting from element 1. 6 BEGIN GETIN DUP 7 ACHECK -- flag Checks ARRAY for proper ascending numerical order. If ARRAY is in the A. 0= IF 1 ELSE ۸ 9 AREVERSE 1 HOVES +! ACHECK 10 THEN proper order, ACHECK returns a true UNTIL CR "You made " NOVES @ 11 flag. . " reversals." CR 12 λ. . Care to play again? " Y/N Om 13 CR REVERSE --UNTIL 14 The game definition. Uses all pre-CR ." Thanks for playing REVERSE... " CR CR ; viously defined words to play the game 15 :\$ of REVERSE. fig-FORTH Version 1.15 N. Burton

```
ok
```

1

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 micro-motion (c) FORTH-79 Version 1.2 to be run on a 48K *Apple II.

Therefore, the following words are non-standard but included in the 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 4th (pun) row 10th column of screen.

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

In closing, I wish to thank Bill Ragsdale for his gracious support and I especially acknowledge the incredibly patient treatment I received from Phil Wasson of 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)

*Apple is a registered trademark of Apple Computer, Inc.

SCR#51

: HOWTODI HOME (31 GAME-TONY LEWIS) . : HOWIGE: HOME (31 GAME-TONY LEWIS)." 31 GAME BY TONY LEWIS)." '31' IS PLAYED WITH A DECK OF 24 CARDS CONTAININING ONLY THE ACES THRU SIXES. EACH OF THO PLAYERS ALTERNATELY DRAWS CARDS FROM THE DECK. ONE CARD AT A TIME. A RUNNING TOTAL IS KEPT OF THE COMBINED"

SUM OF THE CARDS DRAWN. THE PLAYER WHO ARRIVES AT THE SUM OF 31 EXACTLY WINS. THE NETWER PLAYER CAN HARE 31 EXACTLY, THEN THE PLAYER WHO MUST GO OVER 31 LOSES' THE GAME MAY APPEAR TOO EASY, BUT IT IS DECEPTIVE, WHEN LOR IF?3 YOU HAVE"

WON THREE GAMES, TRY TO BEAT THE PROGRAM FOR 'THE BIG BET' BY TYPING IN 'B' RATHER THAN Y OR 'N WHEN 'NEW GAME?' (OMES UP, THE 'BIG BET' IS A TWO GAME SERIES, YOU GO FIRST IN GAME 1 AND

SCR#50

(MORDS OF WISDOM JI BY TONY LEWIS) (THE 'ANSWER' PAGE IS NEXT. IT DOESN'T REGUIRE ANY SFILL TO FIGURE OUT WHAT THE CONSTANTS REALLY ARE 'THEY ARE ENCODED CONSTANTS REALLY ARE THEY ARE ENCODED SU THAT YOU CAN ENTER AND COMPILE THE GARE WITHOUT DISCOVERING ITS PRINCIPLE. REMEMBER. THE PURPOSE OF THIS PROJECT MAS TG GET YOU TO FIRST EXAMINE THE GAME BY PLAYING IT, THEN FIGURE OUT HOW TO APFRUACH THE PROBLEM OF PROGRAMMING IT, AND FINALLY GO BACK AND COMPARE YOUR AND FINALLY GO BACK AND COMPARE YOUR METHODS TO MINE. THE GAME IS AMUSING AND IS A LITTLE KNOWN CINCH BAR BET. IF You take the time to enter it onto your Forth Disc. You should have fun both ANALYZING IT AND THEN ENTERTAINING LOR HUSTLING) FRIENDS AND FAMILY WITH IT. DESIGNES HER DEAVING AT A BAR YOU MUST USE A REAL DECK OF CARDS AS IT MOULD PROBABLY TEND TO DISCOURAGE WAGERING IF YOU SHOULD BRING YOUR 'HICRO' WITH YOU.)

SCOMST

LEWES. (ENCODED CONSTANTS 31 BY TONY LEWIS) (NOTE: THESE CONSTANTS ARE USED ONLY TO CONCEAL THE SOLUTION OF THE GAME, NOT TO MAKE THE CODING HARD TO FOLLOW!)

O CONSTANT EL O CONSTANT K2 CONSTANT 13 O CONSTANT 14

HEX : CODECONS CD EF 8 AB BC 8 DC FE 8 BA CB 8 CE ED 8 AC BD 8 EC DE 8 CA DB 8 4108 ка 1 К2 1 К3 1 К4 -46C7 -3FB6 -

```
DECIMAL
```

SCR#54 SETUP AND UTILITY WORDS 31-TONY LEWIS:

CREATE DECH 0, 4, 4, 4, 4, 4, 4, 4, 0, VARIABLE CARDSUM VARIABLE GAMESHON

NEWGAME (FIRST, NEW DECK) 7 1 DO I 2% DECK + 4 SWAP ! LOOP (CARDSUM ' HOME 4 CV t NEWGAME

SHOWDECKSUM CR CR

- SHUMDELS.SUM CA CA "THE DECK NOW CONTAINS " 7 1 DO 1 (NOT J!!) 28 DECK + C0 (DUCK O DO) IF 1+ 1 CA DO J . LOG ELSE DROP DUP LOOP

 - THEN 12 CH LOOP
- . " THE RUNNING TOTAL IS " CARDBUN @ . 1
- : BADPLAY (FLAG BAD PLAY) O CR CR ." BAD TYPE-IN" SHONDECKSUM 1

SCR055 (UTILITY WORDS CONT. 31 BY TONY LEWIS) : UPDATEDECKSUM (1 TO 6 NOW ON STACK) UPDATEDELKSUM (1 TO 6 NOW ON STACK) DUP 21 DUP DECK + CƏ DUP (ANY LEFT?) IF 1- (UPDATE DECK) SHAP DECK + C' CARDSUM CƏ + CARDSUM (' (NEN SUM) 2 (CARD-IN-DECK FLAG) ELSE DROF DROP DROP () TICM -THEN : SI, KW'io MALLALINE MORDS 31 BY TONY LEWIS) PLAYERNOVE CR CR " TYPE IN LARD 1 - 6 " KEY CR 48 - (FRCH ASCII) DUP DUP

(CHECK VALID ENTRY)

- IF OL (CARD 1 - 6 ?) 1F
- UPDATEDECKSUN SHAP . DUP
- IF UPDATEDELASUM SMAF HOME ." YOUR CARD MAS A ". L OF (IS CARD IN THE DECK") IF DROF (FLAG BAD PLAY) O CR CK ." CARD NOT IN DECK" SHOMDECKSUM

- THEN
- ELSE DROP DROP DROP BADPLAY THEN
- 60.6857 (MAINLINE WORDS CONT. 3) BY TONY LEWIS) HYCARD

 - - UM (T FLAG ON VALID CHOICE) IF DROP 1 DUP UPDATEDECI.SUM
 - 16
 - ELSE DROP 2 OF DATEDECT SUM THEN & FLAGECARD, SO NO DROFT

THEN THEN IS " . SHOHLECKSUM

501408

MAINLINE WORDS CONT. 31 BY TUNY LOWIS/

x x is in CARL

- H SIN PLAYERMUNE
- (ILLEGH. PLAY LEAVES () ON STACK) UNTIL SHOWDECTSUM () (PLAYER MODE LAST PLAY) ;
- 1 YOUWIN OR OR SETING ." DH RHIS' YOU WIN." SETNORH GAMESHUN CH I+ GAMESHON L' I
- YOULOSE OR OR SETINV ." YOU LOSE, BETTER LUCH MEAT TIME." SETNORM 1

SCR#5-

- (MAINLINE WORDS LUNT. 31 BY TONY LEWIS) : NORMAL31 HOME 7 CV
- " DO YOU MANT FIRST PLAY" TYPE Y OR N." CR LEY 70 = (N) 1 (1ST MOVE) MYCARD ELSE YOURCAND I۲
- THEN U SHAP & SET UP LOOP

- THEN C UNIT BEGIN IF C TRUE FLAG SET ON MY(ARD) 31 CARDSHM CO IF YNUMIN IF C SET LOOP EATTY IF YNUMIN IF C SENER CO *
 - ELSE 31 CAROSUM CO * IF VOULOSE 1+ ELSE YOURCARD 0
 - THEN
- HEN
- ELSE (RETURN FROM YOURGARD)
 - 31 CHRDSUM La
 - IF YOULUSE 1+
 - ELSE 31 CARDSUM CA = IF YORWIN 1+ ELSE 0 MYCARD 0 THEN C NOT 1ST HOVE
- THEN

```
THEN
UNTIL O & LUDP BACK IN MAINSLY E
```

Fd

5

•

SCR#60 MAINLINE WORDS CON1. 31 BY 10NY LEWIS) MYBIGBETI CR CR K1 DUP CARDSUM C0 + k2 - k1 MOD -DUP UPDATEDECKSUM U. IF DROP K3 DUP UPDATEDECKSUM DROP THEN ." MY PLAY IS " . SHOWDECKSUM 3 & : MYBIGBET2 CR CR IF DUP UPDATEDECKSUM DROP ELSE KI CARDSUM CO K2 - FI MOD - DUP UPDATEDECKSUM OF IF DROP KA DUP UPDATEDECKSUM DROP THEN .. MY PLAY IS " . SHOWDECKSUM 3 : SCR#61 MAINLINE WORDS CONT. 31 BY TUNY LEWIS) RIGBETI YOURCARD BFGIN IF (TRUE FROM MYBIGBETT) TI CARDSUM CA < TI CARDSUM CA < IF YOUWIN 1 (SET LOOP EXII) ELSE 31 CARDSUM CA = IF YOULOGE 1 ELSE YOURCARD O SHEN. FLEE (RETURN FRUN YOURLARD) SI CARDSUM Car . IF YOULUSE 1 ELSE 31 CARDSUM CO = 10 YOUWIN 1 ELSE MYBIGHETI 0 THEN HEN f her ta UNTIL : SCR#62 MAINLINE WORDS CONT. 51 BY TONY LEWIS) BIGBET2 : MYBIGHET BEGIN IF TH LARDSUM (2) a IF YOULUSE 1 ELSE YOURCARD () THEN ELSE 31 CARDSUM CO -F YOULDSE 1 ELSE O CINUT 1510 MYB16BET2 O THEN THEN UNTIL : SURMAS MAINLINE WORDS CONT. 31 BY TONY LEWIST : BIGBET 5 HOME WEICOME TO 'BIG BET', THE FINAL PHASE OF THE 31 GAME, IWO GAMES WILL HE PLAYED." (R. "YOU WILL GO FINST IN GAME TWO, GOUD LOUP." REISHETE LE CR ." HIT ANY FEY AND I WILL HEGIN GAME 2." LA FEY DROF NEWGAML SETELASH ." BIG RET GAME 2 THE FINALE" CR CR SETNURM 816BE12 WELL DID YOU HAVE THE CORRECT ANALYSIS? IF SO, THEN SEE IF YOU CAN FIGURE OUT" . Fr WHO WINS WITH A IST CARD OF ONE ON THU. IT'S A TOUGH COMBINATORIAL PROBLEM """ I (SET FINAL EXIT IN MAIN31) : ""

SCR#64 (PLAY THE GAME OF 31 BY TONY LEWIS) 1 MAIN31 (LOGIC SHELL) O GAMESHON ' BEGIN CR CR . "NEW GAME? " . "TYPE Y OR N OR B(BIG BET)." CR CR KEY DUP 78 - (CHECK FOR N) (FALSE LEAVES 76 ON STACK FOR 'UNTIL') TOTOTOTOT IF NEWGAME 66 = (D) IF GAMESHUN C9 2 > IF BIGBET ELSE HOME O ." YOLI HAVE WON " GAMESHON CO DUP . ." GAME" 1 " IF ." ." ELSE ." S." THEN CR ." YOU MUST WIN 3 GAMES TO " ." PLAY 'BIG BET'." THEN ELSE NORMAL 31 THEN UNTIL : 1 31GAME HOWTO31 CODECONS MAIN31 : CODECONSA MAIN31 NEW GAME? TYPE Y OR N OR B(BIG SET). DO YOU WANT FIRST PLAY? TYPE Y OR N. MY PLAY IS 2 THE DECK NOW CONTAINS 1 1 1 1 2 2 2 3 3 3 3 4 4 4 4 5 5 5 5 THE RUNNING TOTAL IS 2 6665 TYPE IN CARD 1 -YOUR CARD WAS A 3 THE DECK NOW CONTAINS 1 1 1 1 2 2 2 3 3 3 4 4 4 4 3555 THE RUNNING TOTAL 15 5 MY PLAY IS 5 THE DECK NOW CONTAINS $\begin{array}{c}1&1&1&1\\2&2&2\\3&3&3\end{array}$ 4 4 4 4 5 5 5 THE REPAILING TOTAL 15 10 6000 LYPE IN CARD 1 - 6 YUNH CARD WAS A P THE DECK NOW CONTAINS $\begin{array}{c}1&1&1&1\\2&2&2\\3&3&7\end{array}$ 4 4 4 4 5 5 5 THE RUNNING TOTAL IS 16 n 6 6 MY PLAY IS 1 THE DECK NOW CONTAINS $\begin{array}{c}1&1&1\\2&2&2\end{array}$ 3 3 3 4 4 4 4 5 5 5 THE RUNNING TOTAL IS 17 666 TYPE IN CARD 1 - 6 YOUR CARD WAS A 3 THE DECK NOW CONTAINS $\begin{array}{c}
 1 & 1 & 1 \\
 2 & 2 & 2 \\
 3 & 3
 \end{array}$ 4444 THE RUNNING TOTAL IS 20

HY PLAY IS 4 THE DECK NOW CONTAINS $\begin{array}{c}1&1&1\\2&2&2\\3&3\end{array}$ 5 5 5 THE RUNNING TUTAL IS 24 ... TYPE IN CARD 1 - 6 YOUR CARD WAS A S THE DECK NOW CONTRINS $\frac{111}{222}$ 4 4 4 THE RUNNING TOTAL 15 29 MY PLAY 15 2 THE DECK NOW CONTAINS 2 2 3 3 4 4 4 5 5 THE RUNNING TUTAL 15 DE 6 6 8 YOU LUSE. BETTER LUCK NEXT TIME. NEW GAME ' TYPE Y OR N DR H(HIG BET). YOU HAVE WON & GAMES. YOU MUST WIN I GAMES TO FLAY BIG BETT. NEW GAME? TYPE Y OR N OR BUBIN BET . Ut. FORTH CLASSES LAXEN AND HARRIS, INC. 24301 Southland Drive Heyward, CA 94545 (415) 887-2894 Introductory classes Process control Applications programming Systems level programming GREG STEVENSON Anaheim, CA (714) 523-4202 Introductory classes KNOWARE INSTITUTE OF 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

SIMULATED TEKTRONICS 4010 GRAPHICS WITH FORTH by Timothy Huang Portland, OR 97211

In this article, I 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, I had access to a very little known, but well built microcomputer -- MX 964/2 by Columbia Data Products, Inc. of Maryland. This little machine has two 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 x 256 bit mapped graphics. It also includes a 9" CRT, 2 double density drives, keyboard, 4 serial ports, and 4 parallel ports. Its all in one piece. It boots up with whatever operating system is on the disk after powered up and the carriage return key has been pressed. Beautiful isn't it?

However, there is a big problem, as with most microcomputer companies, the instruction manuals are terrible. And I mean terrible! Let me just give you one 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. 512×256 bit mapped graphic is the best that can be expected under the price allowance. There are quite a few well known microcomputers on the market claiming High Resolution Graphics. But those High Resolution ones are just like a big blob compared with the individual pixel that bit mapped. So, I have a nice machine with all the fancy graphic capabilities, but lacking the key to open it. Anxiety mounts up quickly.

I have a friend who's an excellent 8080/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

15

Screen # 10 9 (Video controls for Columnia MI764 12/09/81 > TDH 1 FORTH DEFINITIONS DECIMAL COTOXY (x y ---) 1 0 MAX 23 MIN 33 - SWAP 3 a MAX 79 MIN 33 + 4 IS EMIT ENIT EMIT . 5 4 CLR-VIDEO 24 EMIT . HOME 25 EMIT . 2 CLR CLR-VIDEO HOME . . CLEARSCREEN CLR 8 : LT-C & ENIT UP-C II ENIT . DW-C 10 EMIT . :0 . BELL 7 EMIT . RT-C 12 EMIT . 11 12 CLREOS 23 EMIT . CLREOL 22 EMIT . 13 CLALINE 21 EMIT . 14 15 Screen # 11 12/07/80) TOK 0 - Graphic Package - 1 : 2 O VARIABLE X O VARIABLE Y O VARIABLE L S VARIABLE TH 3 0 VARIABLE XI O VARIABLE Y: GS 29 EMIT i vestor : : IT 12 ENIT . ESC 27 EMIT 5 CAN 21 EMIT (non-grafie) . US 31 EMIT (Alpha) . 4 EM 25 EMIT (clear video memory) 7 BLACK BSC 127 ENIT . WRITE ESC 97 EMIT . 8 , 10 11 12 12 14 15 Screen # 12 TDH 12/09/80 0 (Giaphic Package - 2 VECTOR (* - Lo, HL) 1 2 32 /MOD 1023 AND 1 4 (Lox, MLX --) ICEN ٩. ENIT 44 + EHIT . 32 + (LY, HX - Ĵ 7 YCEN ENIT ENIT . 32 + , PRE-OUT (N.M --) VECTOR YGEN VECTOR IGEN . 10 11 12 PAGE (enter alpha from vector) 13 ESC II . :4 15 Screen # 13 12/09/80 TDH , 3 a (Graphic Tackage 1 2 INIT PAGE 63 32 EHIT 76 ENIT 32 EKIT 64 EHIT 3 4 ENDRAW 5 QUIT . 4 1 DRAW (4 9 ---) . SWAP PRE-OUT SWAF DUP I : DUP Y I 9 10 11 CHOVE & a y --- > 65 DRAV . 12 13 14

PORT

```
Screen # 14
 0 - Graphic Package
                          TDH
                                                  12/07/80 1
 1
 2
     RDRAW ( Relative DRAW )
           Y Q + SWAP X Q + SWAP
                                          DRAV
 5
     RMOVE ( Relative MOVE )
                   RDRAW
          GS
 4
 7
     ACURSOR
                   Calphaourson & y --- ;
 8
                   DRAV
 9
           35
                          20
:0
11
     SCROLL
                   CAN UR ." press FUNC 4 3 keys
12
     LINE-ERASE
: 3
           BLACK
                  X P
                          Ye
                                  PRE-OUT .
14
15
Screen # 15
 0 - Graphic Package
                           - 0
                                          TDB
                                                  11.09/80 /
  1
 2
      INITO
           INIT ROT L & DUP Y SWAP DUP X DWAP OWDVE .
 3
      SQUARE
                   ( 1 x y --- )
  4
                   X & L & + OUF X
           INITO
                                    Y & DRAV
                                                  : ---. .
 5
                                                  V UF -
 4
                   X O V O L O + DUF Y . DRAW
                   X . L . D . DUF X . Y & DRAW
 2
                                                  A 1 --- /
                   X Q Y Q L Q - DUP Y . DRAW
 8
                                                  COWN .
      ERASESGU
                   s 1 x y --- .
 10
           INITO
                  X @ L @ + X 1 LINE-ERASE
 11
                                                    ---; ;
                   Y . L . . Y ! LINE-ERASE
                                                    82 J
 12
                   X 9 L 0 - X E LINE-ERADE
                                                     ---
 13
                   Y & L & - Y - LINC-ERACE
                                                  C DOWN .
 14
 :5
Screen # 16
  6 - Graphic Fackage
                           - 0
                                           10h
                                                  12/20/26
  1
      INITI 6 advance x & v, check range /
            1 9 X 9 X1 9 + BUP 512 SUE 5 ( IT BROE DROP DROP C LUT
            CLSC 1523 1 IT DROP DROP 1023 DUE THEN THEN X
  4
           Y . YC . DUP DUP DUP C C IT DROP DACP DACT & DU.
  5
            ELCE 799 > IF ORGP DROP 79% DUP THEN THEN T
                                     UNDED DI DI 178
      INIT2 L @ X @ I #
  3
 :0
      MANYDQUARES ( 1 2 y tm st 91 ----
           INITS SQUARD TH @ : - C BU INITE SAVARE LOOP
 11
 12
                   1000 3 50 1007
 : 3
      DELAYED
                                   - 10 milisectnis - -
                   11
      2DELAY
```

(we at least knew that the graphic part simulates Tektronics 4010), he spent a whole week just trying to draw one mere aquare along the edges of the CRT. Seemingly it would be an easy job, but even so it never came near to what he would have liked. Later on, I spent a couple of weeks twiddling with Microsoft BASIC compiler and it also produced lousy results.

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

By now, I was very desperate to get it going. Equipped with the FORTH power and the poor manual, I set forth to try the graphics again. Again, I sought help from a friend who works for Tektronics and is experienced with FORTH. With FORTH, the whole task turned into a very simple job, compared to the previous attempts we had with the assembly and BASIC. Thus, now 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 as 779. Screen 12 to 14 defines the basic words, which draw the line, move the cursor, relative draw and move. Screen 15 defines the words to draw a square and the erasing of it. Screen 16 lets me draw many squares.

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



15

A VIDEO VERSION OF MASTER MIND David Butler Dorado Systems

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

The program contains many of the functions found in video editors, including cursor management and character collection. The sequence of this computer version of the game is as follows: After displaying the directions, the program prompts the player to enter his skill level. Then a 'secret code' is generated with the help of the player tapping the space bar. The screen is cleared, and a 'mask' of the Master Mind playing board is displayed. The cursor lands at the location where the player is to begin entering his guess. The program retains control of the cursor, responding to the player's key strokes. Backspacing and tabbing are allowed, enSCR # 18 O (Master Mind in Forth by David A. Butler -DAB-17nov80) 1 --> David A. Butler 2 з 33300 Mission Blvd 4 Apt 126 Union City, CA. 94587 (415) 487-6039 5 6 7 8 0 ##### A note about style: If there is any, it is an accident. 10 This was my first application in Forth, so it may lack 11 some elegance. 12 13 ##### Requirements: A video display 80 x 24 characters, 14 curson addressing and clear screen 15 functions. SCR # 19 0 (Master Mind -notes-DAB-17nov80) 1 --> 2 This is an implementation of Master Mind by Invicta. 3 The same is very popular because it is easy to learn and a 4 challense to play. There is a bit of luck to it, but it is 5 mainly an exercise in logical deduction. A "secret" code is 6 generated, and it is "cracked" by analyzing a set of clues. 8 Those familiar with the original board same will have no 0 difficulty adjusting to the computer version. To newcomers, 10 follow the directions carefully and you will have it in no 11 time. The Forth version is functionally identical to the 12 board version. It is written in fis-Forth, and has been run 13 successfully on 6502, 8080, Z80, and 68000 processors. It 14 is a good demonstration program as well as an enjoyable same. 15 SCR # 20 DAB-17nov80) O (Master Mind set up some variables 2 : TASK ; (FORGETTABLE MARKER) 3 4 O VARIABLE COLORS 28 ALLOT COLORS 30 BLANKS 5 6 0 VARIABLE SCODE 2 ALLOT 7 0 VARIABLE SECRET 2 ALLOT 8 0 VARIABLE BLACKER 0 VAR O VARIABLE GUESS 2 ALLOT O VARIABLE WHITER 9 6 VARIABLE #COLORS 10 23 VARIABLE CUR.COL 11 3 VARIABLE CUR. ROW 12 1 VARIABLE XLOC 1 VARIABLE YLOC O VARIABLE DONE 13 --> 14 15 SCR # 21 DAB-17nov80) set up - cont. O (Master Mind ." YELLOWRED BLACK GREEN WHITE BLUE * 3 2 : C. CONSTANT 4 0 VARIABLE COLOR.KEY 6 ALLOT ("colors" table) 5 $\bar{6}$ (Use the sum of the ASCII code of the first 3 letters) 7 (i.e. BLUE = "B" + "L" + "U" = 66 + 76 + 85 = 227) 8 219 COLOR.KEY 1+ 9 234 COLOR.KEY C' 10 207 COLOR. KEY 2 + C! 222 COLOR. KEY 3 + C! 11 232 COLOR. KEY 4 + C! 227 COLOR.KEY 5 + C! 96 COLOR. KEY 6 + C! 12 13 14 O VARIABLE WATTEMPTS (used to keep score) 15 -->

```
SCR # 22
  O ( Master Mind prompt and randomize DAB-17nov8C
1 ( These definitions set the random values for the same )
                                                            DAB-17nov80 )
  2
  3 : NEWCOUNT ( ECOLOR# + !] ) DUP #COLORS & C
         IF 1+ ELSE DROP 1
  4
                                         THEN 1
  5
  6 : RAND 1 BEGIN NEWCOUNT ?TERMINAL UNTIL KEY DROP :
                          ." To randomize, tap space bar 4 times."
  8 1 ASK FOR RANDOM
         4 0 DO RAND I SCODE + C! LOOP CR 1
  9
 10
 11 : ASK.FOR.LEVEL
        CR . " Level 1 or 2 ? " KEY DUP EMIT KEY EMIT
 12
           50 = 1F
                       7 #COLORS ! ELSE & #COLURS ! THEN CR +
 13
 14
 15 -->
SCR # 23
  O ( Master Mind
                          translate color to numeric
                                                              DAB-17nov80 )
  1
             .FIND ( LCOLOR#) ----U) TYPES COLOR FROM #)
1 - 6 • 7 C.CONSTANT 3 + + 6 TYPE 1
  2 : COLOR.FIND
  2
  4
  5 : TRANSLATE.CODE
              ( converts color # from SCODE to COLOR.KEY )
  6
  7
               ( numeric value in array "SECRET" )
  8
    4 0 DO SCODE I + CE 1 - COLOR.KEY + CE SECRET
  0
         I + C! LOOP ;
 10
 11
 12
 1.2
 14 : J REAR RECOMPILED R SWAP DR SWAP DR SWAP DR :
 15 -->
SCR # 24
  0 ( Master Mind cursor motion
1 ( Of course, CRT dependent. Here is Heath;
                                                             DAB-17nov80 )
                                                                           •
                              ( *** start CRT dependent words *** )
  3 : CURSOR ( [Y] [X]---[] ABSOLUTE CURSOR POSITION )
         31 + SWAP 31 + 89 27 EMIT EMIT EMIT EMIT :
  4
  5
  6 : CLEAR ( CLEAR CRT SCREEN ) 27 EMIT 69 EMIT ;
 8 : HOME ( PUT CURSOR AT HOME POSITION ) O CUURSOR :
9 ( *** end of CR3 dependent words *** )
10 : SHOW.COLORS ( DISPLAY COLOR CHOICES )
       7 1 DO I 2 + 58 CURSOR I COLOR.FIND LOOP
 11
      #COLORS @ 7 = IF 9 57 CURSOR ." \langle BLANK \rangle" FLSE THEN 12 58 CURSOR ." TAB between colors,"
13 58 CURSOR ." RETURN to set clues." ;
 12
 13
 14
 15 ---
SCR # 25
                                                             DAB-17nov80 )
  0 ( Master Mind
                       board layout mask
  1
  2 : BAR ." :" : : DASH ." _" : ( BOARD SYMBOLS )
  3 : TITLE 21 SPACES
  Δ
         ." ==== MASTER MIND =
  5 : DASHER 2 21 CURSOR BAR 32 0 DO ." ~" LOOP BAR UR 4
6 : CLINE DUP 21 CURSOR BAR 54 CURSOR BAR 4
 8 : SPACER 21 CURSOR ." :_^____^' :
9 : CBLOCK DUP CLINE 1+ SPACER ;
10 : HIDDEN 3 23 CURSOR ." XXXXXX XXXXXX XXXXXX XXXXXX ***
 11
 12 : DISPLAY, BOARD
 13 CLEAR TITLE DASHER HIDDEN 24 3 DO I CBLOCK 2 +LOOP
           SHOW.COLORS :
 14
 15 -->
```

abling the player to keep changing his guese until he is satisfied that it is consistent with the clues he has thus far received. A correct guess is the result of the player's logical deduction (or very good luck) based on his previous clues. The directions on screen 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 earlier. Clues are generated and displayed, indicating to the player how close he is to the solution. The player has ten chances to deduce the secret code.

There are many improvements which could be made to this program to 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 # 26
  O ( Master Mind
                      cursor tracking definitions DAB-17nov80 )
  1
  2 : X XLOC @ : : Y YLOC @ :
  3
  4 : XBUMP X 52 =
5 IF 23 DUP CUR.COL ! XLOC !
        ELSE 1 XLOC + X CUR.COL @ 8 + =
IF X CUR.COL ' THEN
THEN ;
  6
  7
  8
  \mathbf{O}
 10 : UNBUMPX X 23 = IF 52 XLOC ! ELSE -1 XLOC +! THEN ;
 11
 12 : TAB CUR.COL @ 47 =
13 IF 23 CUR.COL '
14 ELSE & CUR.COL +
 15
       THEN
              CUR.COL @ XLOC ! DROP Y X CURSOR ; -->
SCR # 27
  0 ( Master Mind character collection/editing DAB-17nov80 )
1 : BACKSPACE X CUR.COL @ =
       IF DROP
  \mathbf{2}
        ELSE UNBUMPX Y X CURSOR SPACE Y X CURSOR DROP
32 COLORS X + 23 - C'
  з
  4
  5
        THEN :
  ۴
  7 : PROCESS ( [CHAR] -- [] PROCESSES CHAR, MANAGES (URSOR )
       DUP EMIT COLORS X + 23 - C! XBUMP Y X CURSOR ;
  8
  Ċ)
 10 : GET.CHAR
                 KEY DUP 127 =
       IF TAB ELSE DUP 9 =
IF TAB ELSE DUP 13 =
IF 1 DONE ! DROP
 11
 12
 13
            ELSE PROCESS THEN THEN THEN ;
 14
 15 -->
SCR # 28
                                                       DAB-17nov80 )
  O ( Master Mind
                        suess / row section
  1
  3
           TAL 2 • 3 + DUP YLOC ! CUR.ROW ! 23 23 XLOC
! CUR.COL ! Y X CURSOR
  4 : INITIAL
  5
           30 0 D0 32 1 COLORS + C! LUOP ;
  6
  7
  8 : GET.COLORS INITIAL O DONE ! BEGIN GET.CHAR DONE @ UNTIL ;
  o
 10 : PARSE.GUESS 4 0 DO I 8 . COLORS + C#
          I 8 * COLORS 1+ + C@
I 8 • COLORS 2 + + C@
 11
 12
          + + I GUESS + C! LOOP ;
 13
 14 -->
 15
SCR # 29
                                                         DAB-17nov80 )
  O ( Master Mind
                       Clue seneration
  1
  2 : CLUE.CHECK
      O BLACKER 1 O WHITER ! ( INITIALIZE COUNTS )
  з
  4
       4 0 00
                             GUESS 1 + C@ = ( CHECK FOR DIRECT HIT )
  5
           SECRET I + Ce
                1 BLACKER +1 O I GUESS + C!
           IF
  E.
        THEN LOOP
  7
                                                ( IF NO HIT )
                   GUESS I + COE O > IF
  8
        4 0 DO
  \mathbf{G}
           4 0 D0
 10
           IF 1 WHITER +! 1 I GUESS + C! LEAVE
THEN
           GUESS I + CE SECRET J + CE = ( CHECK FOR WHITE )
 11
 12
 13
          LOOP THEN
 14
        L00P :
 15 -->
```

5

```
SCR # 30
  O.C. Master Mind
                                                          DAB-17nov80 )
                                                                                       ANNOUNCEMENTS
                          present clues
   : GIVE.CLUES PARSE.GUESS CLUE.CHECK
                                                                             NEW JERSEY FIG CHAPTER BEING
  2
        Y 1 CURSOR BLACKER @ . . " BLACK
  з
                                                                             FORMED
                      WHITER @ . . " WHITE "
  4
                                                    1
  5 : UNMASK 3 23 CURSOR
                                                                             Interested parties should contact:
        4 0 DO I SCODE + C@ COLOR.FIND I 3 =
  6
                                                                             George B. Lyone
         IF ."
                " ELSE ."
                              " THEN LOOP 23 1 CURSOR 1
                                                                             280 Henderson St.
  8
                                                                             (212) 696-7606 - days
  9 : ?AGAIN 20 58 CURSOR ." TYPE MASTER TO" 21 58 CURSOR
10 ." PLAY AGAIN." UNMASK 23 1 CURSOR ;
11 : LOSER 16 58 CURSOR ." NICE TRY BUT" 17 56 CURSOR
12 ." NO CIGAR." ?AGAIN :
13 : WINNER 16 58 CURSOR ." PRECISELY. " #ATTEMPTS ?
14 ." TRYS." ?AGAIN :
15 -->
                                                                             (201) 451-2905 - eves
                                                                             BOSTON FIG CHAPTER SEEKING
                                                                             MEMBERS
                                                                             Interested parties should contact:
                                                                             R. I. Demrow
                                                                             P. O. Box 158, Blv. Sta.
SCR # 31
                                                                              Andover, MA 01810
                                                          DAB-17nov80 )
  O ( Master Mind Directions to Player
                                                                             (617) 389-6400 x 198 - work
  1
                                                                             (617) 664-5796 - home
  2 : DIRECTIONS CLEAR CR CR CR CR CR
3 10 0 D0 LOOP ." Welcome to MASTER MIND." CR CR
                                                                             MOUNTAIN
                                                                                         WEST FIG CHAPTER
    ." The object of Master Mind is to break the secret code."
                                                                             ORGANIZING
  5 CR ." The computer will pick the secret code, and you must"
    CR ." figure it out. Two kinds of clues are given:" CR
  6
                                                                             Interested parties in the greater Salt Lake
    CR ." (1) # BLACK means that you have # pess correct " CR
  7
                                                                             City area should contact:
  \otimes
                           in both color and position." CR CR
                                                                             Bill Haywood
        ." (2) # WHITE means that you have # Pess of the " CR
  \odot
                           correct color that are incorrectly " CR
placed. " CR CR
                                                                             (801) 942-8000
 10
 11
                                                                             TECHNICAL PRODUCTS CO. MOVES
 12 ." Be sure to spell the colors correctly. You may tab amound "
 13 CR ." the 4 Positions until you've make your best duess." CR
 14 CR ." Type [RETURN] to receive clues." CR CR ." Good-luck."
                                                                             New address:
                                                                             P. O. Box 2358
 15
      CR CR ; -->
                                                                             Boone, NC 28607-2358
SCR # 32
  0 ( Master Mind
                                                                             FIG NEW YORK CITY MEETING
                         ++ FINAL ++
                                                          DAB-17nov80 )
                                                                             CONTACT
  1
                DIRECTIONS O 0 #ATTEMPTS !
  R
    : MASTER
                                                                              James Basile
  4
       ASK.FOR.LEVEL
                          ASK. FOR. RANDOM
                                                                              40 Circle Drive
  с;
       DISPLAY, BOARD
                           TRANSLATE.CODE ( Fut UNMASK to debug )
                                                                              Westbury, NY 115900
                                                                             (516) 333-1298
  1.
       0 10 DO
  7
        1 #ATTEMPTS +!
        PARSE. GUESS I GET. COLORS GIVE. CLUES
                                                                              DALLAS-FT. WORTH METROPLEX FIG
  \mathbb{R}
 \sim
        BLACKER @ 4 =
                                                                              MEETING CHANGE
 10
       IF WINNER LEAVE
        ELSE. THEN
                                                                              Meetings now being held at:
 12
       -1 +LOOP
                                                                              Software Automation, Inc.
       BLACKER @ 4 <
                                                                              1005 Business Parkway
 13
 14
       IF LOSER ELSE THEN : MASTER
                                                                              Richardson, TX
 15
                                                                                contact:
                                                                              Marvin Elder (214) 231-9142
              ----
                                                                              Bill Drissel (214) 264-9680
 These are hidden : RED KED BLUE GREEN
during wig. 2
                                      YELLOW
              BLACH
              --------
                                       HITE
              -----
                                      MULTE
              HED RED PLUE GREEN
 4 BLACE & WHITE
             RED BLUE GREEN RED
 1 BLACE - WHITE
             VILLOW GREEN BLACK WHITE
 O BLACK 4 WHITE
 O BLACH 1 WHITE
```

(Snamshot of board after playing Master Hind)

FOR TH DIMENSIONS III/5

TRANSFER OF FORTH SCREENS BY MODEM

Guy T. Grotke Forth Gear San Diego, CA

Here is a simple but hopefully useful set of definitions for serial transfer of FORTH screens between machines. Several of us in the San Diego 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 modern 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 SINPUT. In screen 80, these four words are defined for an APPLE running a serial interface in slot two (driver at \$C200). In screen 58, they are defined for an Ohio Scientific with the normal serial port found in the personal models. These are examples of vectored and flagged I/O redirection.

The remaining definitions should be quite universal among fig (and other) systems. Screen 80 contains all that is necessary to receive screens under the control of the sender. FINISHED and RECEIVE simply redirect input and output. The word P redefines the fig editor word P to do the same thing except with I/O redirection. Note that these three definitions are simple and fool-proof enough that they could be sent to another computer if that computer was first told to accept all input from the serial transfer 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 just 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 transmitter 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 transmitter to wait for the receiver to reply "OK" after each line is sent. SEND is a multi-screen transmitter. Note that the range of screens received and recorded on disk will correspond exactly to the screen numbers sent. If that is inconvenient, a variable containing an offset or starting receiver screen number could be added.

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

SCR	# 58	
i'r	CSerial Screen Transfer sending GTG 7-02-81)
1	HEX	
2	: SOUTPUT 3 2322 C! ; (SEND OUTPUT TO SERIAL + CONSOLE	,
3	: COUTPUT 2 2322 C' ; (SEND OUTPUT ONLY TO CONSOLE	,
4	: SINPUT 1 2321 CH ; (GET INPUT FROM SERIAL)
5	: CINPUT 2 2321 C! ; (GET INPUT FROM CONSOLE)
0	: SOUT SOUTPUT CINPUT ; : SIN COUTPUT SINPUT ;	
- 7	: OK ; : WAIT SIN QUERY ;	
8	: SEND.SCREEN (SCR# ~-> nothing left)
Ÿ	SOUT DUP " COUTPUT LIST SOUTPUT " CR WAIT	
10	10 0 DO 1 SOUT P " I OVER .LINE CR	
11	WAIT CINPUT ?TERMINAL IF LEAVE THEN LOOP ;	
12	: SEND (FIRS1 SCR# / LAST SCR#> nothing left)
1.3	1+ SWAP DO I SEND. SCREEN ?TERMINAL IF LEAVE THEN LOOP	
14	SOUT CR WAIT SOUT ." FINISHED " CR COUTPUT ;	
15	DECIMAL ;5	
SCR	# 80	
υ	(CONSOLE/SERIAL I/O)	
1	FORTH DEFINITIONS HEX	
	: UNLINK FDF0 36 ! FD18 38 ! ;	
	: SOUTPUT C200 36 ! ;	
4		

4 : COUTPUT EDFO 36 !; 5 : SINPUT C200 38 !; 6 : CINPUT ED1B 38 !; 7 8 EDITOR DEFINITIONS 9 10 : FINISHED CINPUT COUTPUT FLUSH; 11 : P COUTPUT P SOUTPUT; 12 : RECEIVE COUTPUT SINPUT; 13 FORTH DEFINITIONS EDITOR 14 DECIMAL

HELP WANTED

Part-time - New York-New Jersey 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.

15 :5

Moderate fees, travel possibilities, hardware experience preferred. Send information or resume to: Max Neuhaus 210 5th Avenue New York, NY 10010 Independent FORTH programmers to implement 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 California to do it.

Contact: Marc Perkel Perkel Software Systems 1636 N. Sherman Springfield, MO 65803 (417) 862-9830 1

4

c

a

a

F

.

11

PRODUCTS REVIEW

SORCERER-FORTH by Guality Software

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

FORTH for the Sorcerer implements Release 1.1 of 8080 fig-FORTH. It includes a full screen editor and input/output routines for the keyboard, screen, and both serial and Centronics printers. The Sorcerer's excellent graphics are also available.

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

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

Quality Software permits--even encourages--users to market application programs incorporating Sorcerer FORTH. They do ask that written permission be obtained 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.

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

A COMPARISON OF TRANSFORTH WITH FORTH Insoft

Medford, OR

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

Floating-point numbers

e.

١Ċ

n-

æ

e.

5

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

Transcendental functions

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

Data structures

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

Strings

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

Disk access and the editor

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

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

Grahics

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

Vocabulary

TransFORTH is a single-vocabulary

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

Compilation and speed

All entries in TransFORTH are compiled directly into 6502 machine language for greater speed. No address interpreter is used. Even immediate keyboard entries are compiled before being executed. This means that routines can be tested at the keyboard for speed before being added as colon definitions.

TransFORTH is fast. It is not as fast an integer versions of FORTH, because it handles 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.

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 code.

2. It is easier to add floating point math to FORTH, than to alter Trans-FORTH to use integers for execution speed improvements. Why not both?

3. If the implementor had done his DOS 3.3 interface using the standard FORTH word BLOCK, an immense gain in value would result. Direct access and DOS compatibility.

4. (BUILDS DOES) probably could be added but apparently the implementor doesn't know how or chooses to deprive his customers of this powerful structure. Arrays are definitely not equivalent technically or philosophically.

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

NEW PRODUCTS

FLEX-FORTH

Complete compiler/interpreter, assembler, editor, operating system for:

APPLE II computers	\$25.00
KIM computers	\$21.00

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

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

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

Order from: GEOTEC, 1920 N. W. Milford Way, Seattle, WA 98177. Be sure to specify machine.

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

Enhanced Z80 fig-FORTH implemented for Northstar System enhancements include link fields in front of name for fast compile speed; dynamic vocabulary relinking; case; arguments-results with 'to' variables: and more. 79-Standard package includes easy to use screen editor.

Price: \$75.00

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

TWO NEW PRODUCTS FROM LAXEN AND HARRIS, INC.

Laxen and Herris, 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 Z80 computer system with the CP/M (TM) operating system.

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

This FORTH implementation is compatible with the popular book "Starting FORTH" by Leo Brodie. It is intended to be a companion to the book to aid learning FORTH. It is also a complete environment for developing and executing FORTH programs. It contains:

Compilers

Disk operating system

Full names stored, up to 31 characters

String handling

Enhanced error checking

16-bit and 32-bit integer arithmetic

and input/output

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

This system as supplied runs comfortably in a 8080 or Z80 computer system with at least 32K bytes of RAM memory, at least one floppy disk drive (8" single density, single sided, soft sectored format is assumed), and the "BIOS" part of the CP/M operating system. The use of a printer is supported but not required. This software may be easily modified to use other memory sizes or disk formats. It requires 14K bytes of memory which includes 4K 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 - plus \$2.00 - Postage and Handling

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

2. Learning FORTH

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

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

POLYMORPHIC FORTH Abstract Systems, etc. 1686 West Main Road Portsmouth, RI 02871 (401) 683-0845 Ralph E. Kenyon, Jr.

Product Description: FORTH (Poly-Morphic fig-FORTH 1.1.0). 8080 fig-FORTH 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 fig-FORTH to interface to the PolyMorphic Systems Microcomputer. Sorted VLIST included.

Implementation document available separately. Separate document available for cost of postage. Product data available on PolyMorphic SSSD 5" diskette format. 4 copies sold to date. Price: **\$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).

HEATH HB9 FORTH MCA 8 Newfield Lane Newtown, Conn. 06470

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

MCA FORTH provides the standard FORTH facilities plus the following special features: HDOS file manipulation capability, a control character to restart FORTH (recover from loops), on-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 programming.

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

MCA FORTH is available from MCA on a 5-1/4" disk for \$25 including documentation. Documentation is available for \$4.00. (Conn. residents please add sales tax).

NEW PRODUCTS FROM INNER ACCESS CORPORATION

 Fig-FORTH compiler/interpreter for PDP-11 for RT11, RSX11M 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 $8"\ single$ density diskette only.

Reference Manual for PDP-11 fig-FORTH above. Price: \$20.00

2. Fig-FORTH compiler/interpreter for CP/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" diskettes in 128 byte, 18 sector/track format and 8" diskettes in 128 byte, 26 sector/track (IBM) format.

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

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

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

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

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

3. METAFORTHTM 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 Model II under CP/M Northstar Horizon Prolog Z80

Released on two 5-1/4" diskettes or on one 8" diskette.

Price: \$450.00

c

Þ

4. Complete Zilog (AMD) Z8002 development system that can be run under CP/M or Cromemco CDOS. System includes a METAFORTH^{IM} cross compiler which produces a Z8002 79-Standard FORTH compiler/interpreter for the Zilog Z8000 Development Module. Package includes a Z8002 assembler, a Tektronix download program and a number of utilities.

Released on two 5-1/4" diskettes or on

one 8" diskette.

Price: \$1,450.00

5. Zilog Z8002 Development Module fig-FORTH 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 requirements: 4.75K bytes free RAM above FORTH. CODE9 is distributed as a commented source listing and manual. **Price: \$20.00**

PET-FORTH by

Uy

Datatronic AB Box 42094 S-126 12 Stockholm Sweden (0)-8-744 59 20 Peter Bengtson

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

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

An expansion disk (coming soon) will contain floating point arithmetic including complex numbers, transparent overlay control words for data <u>and</u> 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 MicroMainFrame.

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

Manual Description: 322 pages, including all source code.

Complete introduction to FORTH. Special chapters cover the asembler, <BUILDS and DOES, IEEE handling, strings etc.

Manual is available separately.

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

Product/Ordering Data: Shipped as diskette and an accompanying security ROM, holding part of the Kernel. Currently, there are approximately 75 installations, after 2 months on the market.

Price: \$290.00 Includes diskette, ROM, manual, shipping and taxes.

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

Delivery is immediate.

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

Diskette of FORTH Application Modules

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 application modules is \$75.00 (if other than 8" standard disk, add \$15.00). To order the FORTH modules, write Timin Engineering Company, 9575 Genesse Avenue, Suite E-2, San Diego, CA 92121, or call (714) 455-9008.

AUDIO TAPES OF 1980 FORML CONFERENCE AND 1980 FIG CONVENTION

- 1. FORTH-79 Discussion, 200 min. Price: \$35.00
- 2. Purpose of FIG, 37 min. Price: \$10.00
- 3. Charles Moore, 63 min. Price: \$15.00
- 4. FORTH, Alan Taylor, 47 min. Price: \$15.00

Complete set \$65.00

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

FORTH VENDORS

The following vendors have versions of FORTH available or are consultants. (FIG makes no judgment on any products.)

ALPHA MICRO

Professional Management Services 724 Arastradero Rd. #109 Palo Alto, CA 94306 (415) 858-2218

Sierra Computer Co. 617 Mark NE Albuquerque, NM 87123

APPLE

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

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

George Lyons 280 Henderson St. Jersey City, NJ 07302 (201) 451-2905

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

CROSS COMPILERS Nautilus Systems

P.O. Box 1098 Santa Cruz, CA 95061 (408) 475-7461

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

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

M & B Design 820 Sweetbay Drive Sunnyvale, CA 94086

Micropolis

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

North Ster

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

PDP-11

Laboratory Software Systems, Inc. 3634 Mandeville Canyon Rd. Los Angeles, CA 90049 (213) 472-6995

OSI

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

Software Federation 44 University Dr. Arlington Heights, 1_ 60004 (312) 259-1355

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

Tom Zimmer 292 Falceto Dr. Milpitas, CA 95035

1802

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

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

TRS-80 The Micro Works P. O. Box 1110 Dei Mar, CA 92014 (714) 942-2400

Miller Microcomputer Services 61 Lake Shore Rd. Natick, MA 01760 (617) 653-6136

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

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

6502

Eric C. Rehnke 540 S. Ranch View Circle #61 Anaheim Hills, CA 92087

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

8080/Z80/CP/M

Laboratory Microsystems 4147 Beethoven St. Los Angeles, CA 90066 (213) 390-9292

Timin Engineering Co. 9575 Genesse Ave. **#E-2** Sen Diego, CA 92121 (714) 455-9008

Application Packages

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

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

68000

Emperical Res. Grp. P. O. Box 1176 Milton, WA 98354 (206) 631-4855

Firmware, Boards and Machines 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 FORTH Products Interactive Computer Systems, Inc. 6403 Di Marco Rd. Tampa, FL 33614

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

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

Consultants Creative Solutions, Inc. 4801 Randolph Rd. Rockville, MD 20852

Dave Boulton 581 Oakridge Dr. Redwood City, CA 94062 (415) 368-3257

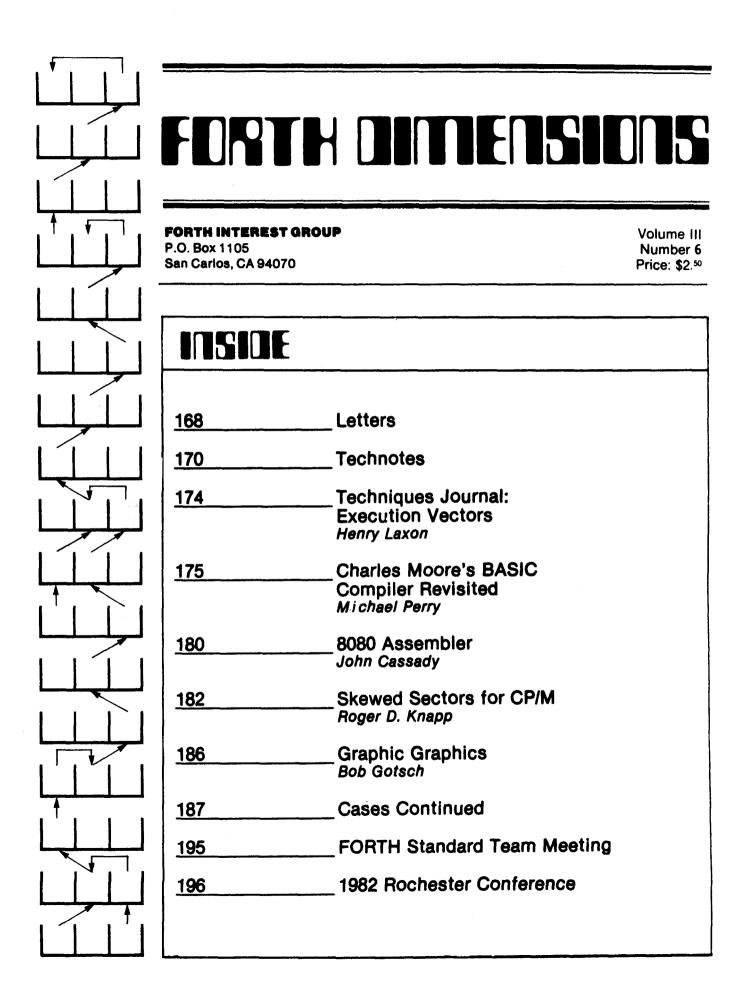
Go FORTH 504 Lakemead Way Redwood City, CA 94062 (415) 366-6124

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

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

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

Microsystems, Inc. 2500 E. Foothill Blvd., #102 Pasadena, CA 91107 (213) 577-1471



FORTH DIMENSIONS

March/April 1982

Roy C. Martens Leo Brodie

Published by Forth Interest Group

Volume III No. 6

Publisher Editor

Editorial Review Board

Bill Ragadale Dave Boulton Kim Harris John James Dave Kilbridge Henry Laxen George Maverick Bob Smith John Bumgarner Gary Feierbach Bob Berkey

FORTH DIMENSIONS solicits editorial material, comments and letters. No responsibility is assumed for accuracy of material submitted. MOST 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 \$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 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 Northern California. 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.

PUBLIC NOTICE

Although the FORTH Interest Group specifies all its publications 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 Talbott Microsystems and the Alpha-Micro Assembly listing by Robert Berkey. Several conference papers have had copyright reserved. The general statement by FIG cannot be taken an absolute, where the author states otherwise.

FROM THE EDITOR

Hi! I'm happy to say that starting with this issue, I'll be serving as regular editor of FORTH 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.

Leo Brodie

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 issue, FORTH DIMENSIONS will give preference to articles that adopt the 79-Standard

Listings which us words that are <u>not</u> 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.

С

24

×

su

21

31

bend

sam

LETTERS

FORTH Application Library

Dear fig,

As distributors in the UK for FORTH inc., with a rapidly growing customer case, we are potentially interested in any application software that is generally useful.

Most of our customers are in the process control/industrial/scientific sectors which, by their nature, require fairly specialized and customized software. Nevertheless, we are sure there are many areas of commonly useful software and that such software would be useful even if only as a starting point or guideline, in order to avoid too much reinvention of the whee!!

Such software might be offered as free and unsupported, at media cost, or as a chargeable product. Whichever way, it needs to have at least some documentation, (i.e., overview and glossary) but it coses not have to be a professional package.

We have an initial enquiry from a user who needs a 3-term controller program for servo control, and some process mathematics for numerical filtering and linear conversion. As he said to us, "surely someone has done this before and written up enough to be useful?". So can you relp? If you're offering something free, perhaps 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 marketable value, we are interested in reloing to create and promote viable packages. We'll not make any firmer plans or suggestions until we hear from you!

> Nic Vine Director COMSOL Treway House Hanworth Lane Chertsey, Surrey KT16 9LA

Benchmark Battles

⊃ear Fig:

te

16

I believe that the primary consideration of an implementation be fluency of $J \approx$, and not speed or size except when specific problems arise. But after reading the "Product Review" in FORTH Dimensions III/1, page 11 and seeing some penchmarks, I couldn't resist trying the same on my own home-brew implementation: 4mHz Z-80, S-100 bus (one wait state on all memory ref's). These are the results I got, plus another column correctng for my slower clock (but not for the

wait state). I guess I designed for speed.

Just want to stick up for the ol' Z-80. If other people can brag about how compact their implementations are, can't I brag about how fast mine is?

brag about it		•
-	Timin	Duncan
LOOPTEST	2.3	2.9
-TEST	5.9	7.4
*TEST	44.0	54.9
/TEST	74.3	88.6
	Bonadio	4=61
LOOPTEST	1.7	1.1
-TEST	6.8	4.5
*TEST	17.5	11.7
/TEST	29.4	19.6

Note

All times in seconds. Each test involves 32767 iterations.

No, I don't use any special hardware. Just the normal Z-80 instruction 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.

> Allan Bonadio 1521 Acton St. Berkeley, CA 94702

Editor's Note:

Here is the code for the benchmarks published in Volume III, No. 1:

: LOOPTEST
7FFF 0 D0 LOOP;
:-TEST
7FFF0D0IDUP - DROPLOOP;
: *TEST
7FFF 0 DO I DUP • DROP LOOP;
:/TEST
7FFF 0 D0 7FFF 1 / DROP LOOP

To "G" or not to "G"

Dear Fig,

I would like to commant on the "Starting FORTH Editor." The "M" command 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 happen), 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 screen being edited should change. M violates this rule, G does not.

One further point: G inserts the new line <u>at</u> 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.

Mike Perry

I agree! 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 same 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:

0 T U This will be the new line zero 0 T X U

The second phrase swaps lines zero and one.--ed.

FORTH in its Own Write

Dear Fig,

:

The two paragraphs below appeared in an 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 lanquage different from others in common use. However, much of it results from its historical development in systems work and read-only-memory-based in 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 readabilitv.

FORTH benefits most from a new, different programming style; techniques blindly carried over from other environments can produce cumbersome results.

It still eludes me as to why people insist on building things into FORTH which are "imports" from other language 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 FORTH should center around removing the cryptological do-dads that are used. For instance, "@" 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 programming 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 prefixed 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 elegant 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.

> J.T. Currie, Jr. Virginia Polytechnic Institute 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.

Minnesota Chapter

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 Northern 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 Commui-Tree 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, turm-key FORTH system that will turn on their machines. We currently have such software for the Apple II, SYM-1, are close on an Osborne-1, 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 (55xxx and 56xxx zip codes, I believe).

Hope to hear from you soon!

Mark Abbott Fred Olson Co-founders of MNfig

Happy to hear about your new chapter: Your mailing 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 those 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 see those handouts from all the Chapters! Write to:

John Cassady 339 15th Street Oakland, CA 94612

Brain-System

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. I am 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 a 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.

I 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 Jerusalem, Israel December 1981

Anyone follow that ?--ed.

TECHNOTES

ENCLOSE Correction for 6502

> Andy Biggs 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, so 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 80 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 revised 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 65F2 WITH 16-51T INDEXING

THE 'Y' REGISTER FORMS THE LOW INDEX BYTE SIACK LOCATION \$1.X FORMS THE HIGH INDEX BYTE THE DASE ADDRESS HELD IN \$4+2. \$8+3 IS ALSO AFFECTED

	. EYTE . WORD . WORD	\$87.'ENCLOSE' L243 *+2	
	LDA JSR TXA SEC	#\$2 \$SETUP	
	SBC TAX	* \$8	
	STY STY	\$3,X \$1,X	: INITIALISE AS BEFORE : Setting Hi index = #
	DEV DEC	SN+3	•
L313	DEC 1Ny BNE	\$1,X XXX1	; PRIME THESE VARIABLES FOR LOOP
	I NC I NC	SN+3 S1,X	: INCREMENT HI ADDRESS ; AND HI INDEX
XXXI	L DA CMP BEQ	(\$N+2),Y \$N L313	; GET CHARACTER FROM INPUT STREAM ; IS IT DE_IMITER 7 ; LOOP IF TRUE
	STY LDA	\$4,X \$1,X	; NON-DELIMITER SO PUT FIRST ; RESULT ON THE STACK
L318	STA LDA	\$5.X (\$N+2),Y	; GET CHARACTER AGAIN
2318	BNE	L327	; BRANCH IF NOT & NULL
	STY STY LDA	\$2,X \$#,X \$1,X	; TIDY UP RESILTS FOR 'NULL' EXIT
	STA TYA CMP	\$3.X \$4.X	: IF FIRST AND LAST INDEXES ARE EQUAL
	BNE	L326 \$1.X	
	CMP BNE INC	\$5,X L326 \$2,X	; THEN ; INCREMENT THIS RESULT
1326	BNE 1 NC JMP	L 326 \$3,X NEXT	
L 327	PHA STY	\$2.X	: SAVE CHARACTER
	LDA Sta	\$1.X \$3,X	; SAVE CURRENT INDEX AS OFFSET TO ; FIRST DELIMITER AFTER TEXT
	INY BNE INC	XXX5 \$1,X	; INCREMENT INDEX
XXX5	INC Pla	SN+3	: AND HI ADDRESS : Recover Character
	CMP BNE	SN 1318	: IF NOT DELIMITER ; THEN LOO?
	STY JMP	SU.X NEXT	ELSE EXIT

1

•

,

÷

.

2

¢,

C e

3

ħ

đ

ţ-

r -

.5

n--

a

A.,

ies.

ta.

875

-.

::

æ-

ner -

۰:

ate ms.

٠...

TRANSIENT DEFINITIONS 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 The word TRANSIENT compilation. moves the dictionary pointer to the "transient area" which must be above the end of the current dictionary. The temporary definitions are then compiled into this area. Next, the word PERMANENT restores the dictionary to its normal location. Now the application program is compiled and the temporary definitions are removed with the word DISPOSE. DISPOSE will take a few seconds because it goes through every link (including vocabulary links) and patches them to bypass all words above the dictionary pointer.

NOTE: These words are written in MicroMotion's FORTH-79 but some non-79-Standard words are used. The non-Standard words have the fig-FORTH definitions. FIRST 1000 - CONSTANT TAREA (Transient area address) VARIABLE TP TAREA TP ! (Transient pointer) : TRANSIENT (--- ADDR) HERE TP & DP I I PERMANENT (ADDR ---) HERE TP ! DP ! ! : DISPOSE (---) TAREA TP ! VOC-LINK BEGIN DUP BEGIN & DUP TAREA UK UNTIL DUP ROT ! DUP O= UNTIL DROP VOC-LINK @ BEGIN DUP 4 -BEGIN DUP BEGIN PFA LFA & DUP TAREA U(UNTIL DUP ROT PFA LFA ! DUP O= UNTIL DROP & DUP O= UNTIL DROP ICOMPILE FORTH DEFINITIONS / (Example) TRANSIENT : CASE : OF ... 1 : ENDOF : ENDCASE ... J PERMANENT : DEMO1 · · · CASE ... OF ... ENDOF ... OF ... ENDOF ENDCASE > TRANSIENT : EQUATE (N ---) CREATE , IMMEDIATE DOES) @ STATE @ IF COMPILE LITERAL THEN : 7 EQUATE SOME-LONG-WORD-NAME PERMANENT : DEMO2 (SOME-LONG-WORD-NAME is compiled) SOME-LONG-WORD-NAME . / (as a literal) (Removes the words EQUATE, SOME-LONG-WORD-NAME,) DISPOSE (CASE, OF, ENDOF, and ENDCASE from the) (dictionary.) (Test DEMO2, it prints a seven,) DEMO2 7 OK

RENEW TODAY!

Fi

NOVA bugs

John K. Gotwals Computer Technology Department South Campus Courts C Purdue University W. Lafayette, IN 47907

I have just finished installing fig-FORTH 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 I 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 $\langle 51 \rangle$ 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:

D

)

BLANKS expects a word address and word 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 NOW!

RENEW TODAY!

Robert L. Smith

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.

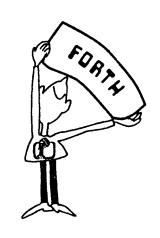
The second point that is mentioned is that the parameter range seems to have a strange asymmetry. When a positive increment is used, the DO-LOOP index I may not reach the specified limit. However when a negative increment is used, the index I may be equal to the specified increment. Users of fig-FORTH 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, FORTHs 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 FORTH Looping Structure." I recommend that readers interested in the topic get a copy of this paper and implement his suggested words. I would like to slightly modify his results for the current discussion. Berkey essentially 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 and n-1 (actually n+1 in Berkey's paper) is crossed dynamically. The implementation appears to be even more efficient than that described by Brodie and Sanderson ("Division, Relations, and Loops," Rochester Conference, 1981). The only apparent disadvantage of the implementation is that the index is computed by addition or subtraction. 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 suggests 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.



Position Wanted

I am looking for a software engineering position with another company 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

9900 Trace

Heinz F. Lenk Loewensteiner Ring 17 6501 Woerrstadt Germany

I have had some trouble getting my 9900 FORTH running.

To ease the finding of errors I wrote a program to display all important vectors (IP, W, CODE, R, SP) and the first 7 stack contents. Even the stack's growing is visible.

I would like to contribute it to you, so you can offer it to all 9900 users with a 100M or similar board.

It was a great luck for the that I did not need the addresses >37C and >37E, and could use it for a branch to the STATUS program. This program is switched off hy the code HEX 455 384 ! and switched on by HEX 457 384 !.

The program list contains the routines for terminal input and output, too.

I hope I can help some people with my program.

		* +10	-61671	4 99(0)			
	#						
					FORT THE 'LEY' AND		
		A 'EMIT'	INSTR	ALL LONS.			
		1					
					HERE IS A PRUBLAM		
					ERG AND THE FERST		
		* SEVEN !					
				OF WENSIED	NER RING 17, 4501 WEDRHSTDT		
		# FIERMAN			12706781		
				-44	******************		
44	FFOO			FFIN)	XOF & VELENCOR		
46	80		DATA	HE NTRY	XUE 1 VECTOR PC		
48	FFOO		DATA	RENTRY FEOD	XDF 2 VECTOR WP		
4A	8E		DATA	ENTISE	XOF 2 VEXTOR FC		
4C	FF20		DATA	-FE20	XER S VECTOR WP		
4E	C8		DATA	MENTIC	XUP 5 VEELON PC		
50			DATA	-FF20	XUP 4 VECTOR WP		
52	03		DUTU	MENTRY	XOF 4 VECTOR PC		

					INTED BY RA		
		# CALL WI					
.			AURG		A 14 A 14 A 14 A 14 A 14 A 14		
80		RENTRY	11	кі, но	tik 1. CROMANDIN		
82	00			***	RETERNER HER - FURT		
86	1F15 16FC		114	R ENIRY	REAL CALL AND TO CALL		
68				#K11,/	EF HOM INSCER		
88			510K		REDEAVER ANTE DSB		
80			RIWE	10			

		I WELTE #	A CHAR	ALLER TO	ERMEN/AL "EMIL"		
		# CALL WI	HI XCH	4141. C			
8E	200	EMITSP	1.1	12, HO	SE. L. CROBOSE		
90	80						
	1010		S\$0		R1S ON		
94		122	115	22	TRANSMER DAMES ENDAX?		
96	16FE		JNE	135			
98		127	110	27	DBR 11		
9A	16FE		JNE.				
90	3218			#R11,8	DUITHT TRANSMIT BUF, EMPTY?		
9E A0		TE22		22 1E22	TREMEMORY DURA REPORT		
A2	16FE		JNE 15	23	HANSHIT SHEFT EMPTY?		
A4	16FC		JNE	1622	TRANSPORT BUSY CLEW CC.		
A6	380		RIM				
-10	200	********					
AB	OAOD			OAOD	CK.EI		
AA	0.00	~*		"(P="			
AE		S 2		" W-"			
Đ4		53		" CODE -	•		
BC		54		" R+"			
C2		55		" SP-"			

• SUBPROGRAM TO OUTPUT A STRING TERMINATED BY >00 • CALL WITH XOP GADRESS, 3 CB CA CC CE DOBB MENTR MOVB #R11+.82 FETCH BYTE JEQ MEXIT EXIT IF ZERO 1302 C282 10FC PRINT ASCII CHAR. JHP MENTR HEXIT 00 RTWP 380 SUBPROGRAM TO OUTPUT A HEX WORD . CALL XOP SOURCE, 4 COUNT D2 201 WENTRY LI ft1,4 D4 4 \$R11,R11 FETCH WORD C2EB HOV D6 D6 846 SRC R11,4 AL IGNMENT DA R11,82 COPY COUD **MNEX** F MOV 242 ANDI R2, 00F00 MASE OUT DE E0 E2 E4 E4 FOO NUMBER? C1 2.3900 282 900 1207 **JLE** NUM R2, 3700 ADJUST LETTER 222 AI EA 200 R2, \3000 ADJUST ASCII ÊA 222 NUM AI CC 3000 E.C. 2082 XOP R2.2 001601 SKC R11,12 SHIFT FO DCB F2 F4 601 DEC R1 FOUNT - L WNEXT 1612 ZERU JNE Ŧ6 380 RIW EXII ********************** ****** . PRINT STATUS PROGRAM & USED FOR DEBUG DURING SET UP SPARE FB 1000 STATUS NOF FA • AB, 3 MSG 71P 200.0 XUP FC FE AA C(H)9 MOV R9,80 COPY 100 640 DECT RO ULD ZIP R0,4 DITENT ZIE XOF 102 2Dem: 1114 213-0 . XOP 5 AL, 3 MSG ZW 106 N 100 COOB NUV RLL, RO 640 2000 104 DECT RO RO.4 100 XLA 201 0 XOP 5 :64.3 MSG CODE IOE 110 64 112 2005 XIN, R5.4 114 2CE0 XCM-6 det. 3 MIG ZR 116 FIC: 810,4 5 C2 118 210494 21.06 2010 MSIG ZSF XOP 11A 110 02 2008 XIN 115 **R0.4** 120 1000 NU # OUTPUT 7 STACES 122 204 11 KI, / COON 124 7 C0€0 MIN 9-31A.RT FEICH STACK START STAX 128 31A R2. 32000 SPACE 120 202 1.1 1,202 2000 MOD FLANC 12E SCC5 STOD XUP R2, 3 2013 XEN #H3,4 PRINT HEX 1.50 CURRENT STACK POINTER? 132 8203 1303 U GEQ R3. 88 SITEXIT 643 DECT R3 NEXT STAD 136 130 COUNT 1 601 DEC - R1 13A 1619 JNE STOUT REPEAT UNTIL ZERO 1 503 455 STEXIT н #Z [1.P#*1 RESUME WORL BY INS # THE URIGINAL DYNAMIC RAM ALLOCATION PROGRAM DOES NOT WURK WITH AN UNTERMINATED DATABUS. THIS 4 LINES SOLVE THE FRUELEM. ADREE 34A FLR 4R5 346 4D5 CLEAK RAM ADDRESS C555 NEW 1K5, 1K5 34C DUMMY 1.4-8 34E JEO. SEARCH JUMP HACK IF ZERD 3:50 1000 NOF THE INNER INTERPETER IS CHANGED TO PRINT ALL
 POINTERS (IP,W,CODERODY,R,SP) AND STACE.
 THE STATUS IS SWITCHED ON BY HEX 457 384
 THE STATUS IS SWITCHED OFF BY HEX 455 384 AURG \37C
 I 1 R7,>FB PC OF STATUS HEX 457 384 1 207 37C 37L 300 F8 C2F9 #ZIP+,ZW HOV C178 MOV #ZW+, ZTEHP POINT TO BODY 382 1R7 BRANCH TO STATUS 384 457 B

A TECHNIQUES TUTORIAL: EXECUTION VECTORS

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 figFORTH EXECUTE operates on code field addresses (cfa'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 perform 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:

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:

L CONTROL-EHIT (CHAR ---) DUP 32 (BLANK) < IF (Control Char?) 94 (^) (ENIT) (Yes, emit en ^) 64 (ABCII A - 1) = (and convert it) THEN (EMIT) a

' CONTROL-ENIT 'ENIT !

Now all regular characters will fail the test, since they will be larger than blanks;

FOR TH DIMENSIONS III/6

Henry Laxen Laxon & Harris Inc. 24301 Southland Drive Hayward, CA 94545

however, control characters will succeed 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 utilities
- 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 definition 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 running, 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 vectors alone. So use vectors sparingly, otherwise you will lose control of the complexity very very quickly.

Having decided to use execution vec-. tors, 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 overhead. Here is another way to accomplish the same thing without having to define a variable. Consider the following:

I DIE (----) EXECUTE: CREATE (----) E'3 DIE , DOES> (----) a EXECUTE ; • IB • 1 5 (PFA ----)

DIE is used to send an error message to the terminal and reset the FORTH and into a clean state. EXECUTE: is a defining word which initializes itself to DIE. but hopefully will be changed later by the user. Words defined with EXECUTE: can be changed with IS as follows:

- EXECUTE: EMIT
- ' (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 can 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 an exercise for the reader to define an IS that may be compiled within : definitions.

Another approach to redefining execution vectors is via the word ASSIGN. It could be defined as follows:

```
1 (ASSIGN) ( CFA --- )
R> 2+ SMAP ! ;
```

1 4951 PM (----)

COMPILE (ABSIGN) [' : CFA @] LITERAL , ; IMMEDIATE

It would be used as follows:

: UPPER-ONLY (----) ['] ENIT ASSIGN DUP 96 (ASCII a-1) > IF DUP 123 (ASCII z+1) < IF 32 -THEN THEN (EMIT) (AS ALWAYS) ;

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.

CHARLES MOORE'S BASIC COMPILER REVISITED

Michael Perry

In this paper I will discuss several interesting features of the "BASIC Compiler in FORTH" by Charles Moore (1981 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 useful 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.

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 discuss 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 inside 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 compilation must be immediate. This use of the FORTH compiler was perhaps my greatest lesson from studying this BASIC compiler. The ordinary FORTH compiler is far more versatile 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 result.

Let's look at an example. The BASIC statement 10 LET X = A + Bwill be compiled into object code equivalent to the FORTH expression $X = A \oplus B \oplus + SWAP!$

where X, A, and B are variables. One of the variables (X) returns an address, the rest return values (with a fetch). The add is compiled after the fetches of the values to be added. The equals becomes the "SWAP!" at the end. Because the source code (in BASIC) is in algebraic notation, and the (FORTH) object code is in reverse pollsh order, some way is needed to change the order of operations when compiling the BASIC program. The mechanism which controls the compilation order is based on the idea of operator precedence, which means that some operators are assigned higher priority than others.

PRECEDENCE

The idea of operator precedence is a prominent feature of most computer languages (FORTH is a notable exception). Operations are not necessarily performed in the order you specify. An example will help. The equation $X = 5 + 7 \cdot 2$ could mean either $X = (5 + 7) \cdot 2$ or $X = 5 + (7 \cdot 2)$, usually the latter. In FORTH this would be $72 \cdot 5 + X$; where the order is explicit. In algebraic languages some method is needed to clarify the order of evaluation of operators in expressions. 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 RUN) the compilation of many words is deferred. This allows the order of words to differ between the source code and the object code. Take '+' as an example. To defer compilation of '+' a new word is created which is immediate (and so executes at compile time). When this new word is executed, it leaves the address of '+' on the stack, and on top it leaves the precedence 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 address 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 consists of an address and a precedence value. If the precedence of the top pair is larger than that of the lower, DEFER does nothing. If the top precedence is less than or equal to the one below, the address 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 started? Essentially, most BASIC keywords (such as LET) execute START with 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 compilation of that statement, because everything has higher than zero precedence.

At the end of each line, RPN is execut ed. It performs a 0 1 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 finish the previous statement) and START (to begin the next).

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, skipping (usually) the following JUMP.

(NEXT) is used for FOR-NEXT loops. It is compiled followed by an absolute address. When executed it takes three parameters from the stack: 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 GOSUB would require another branching primitive, CALL.

ь њ

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 (#S) along with the address (HERE) of the start of that statement. STATEMENT then de-allocates the space used by the literal 10 (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 BASIC programs must be compiled below 9000H, so that all addresses appear to be positive. Here simplicity was chosen over generality.

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: X A @ B @ + 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 1. "INTEGER X" 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 @ after the address, thereby rturning the value when the BASIC program is run. Notice that the equals sign playe SE role in this process; everything is dans by keywords (e.g., LET) and variables.

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 compiler to write compilers for other, more useful, languages such as C. A C compiler which is easy to modify and extend, and just as portable as FORTH is, could actually be It is peaking the second secon

(screens on following pages)

Transportable Control Structures With Compiler Security

> Marc Perkel Perkel Software Systems 1636 N. Sherman Springfield, MO 65803

This article is an 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 ge	ap.

- >RESOLVE Resolves forward branch and leaves a gap.
- <MARK Marks destination of backward branch.

<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.

Example:

:

1

>MARK HERE 0,; >RESOLVE DUP @ 0 ?PAIRS HERE SWAP !; <MARK HERE; <RESOLVE DUP @ NOT 0 ?PAIRS,; IF C, >MARK; ENDIF >RESOLVE; ELSE C3 IF SWAP ENDIF; BEGIN <MARK; UNTIL C, <RESOLVE; AGAIN C3 UNTIL : WHILE IF; REPEAT SWAP AGAIN ENDIF;

```
0 ( Charles Mcore's BASIC compiler, modified for fig-FORTH )
                                                                ( Precedence )
 1 VOCABULARY ARITHMETIC ARITHMETIC DEFINITIONS
 2 VOCABULARY LOGIC VOCABULARY INPUT FORTH DEFINITIONS
 4 : +LDAD BLK 2 + LDAD ;
 5 : (SET#) RL WORD HERE NUMBER DROP :
 6 : (,) S-30 SWAP OVER DABS (* #5 SIGN #2 ;
 7 0 VARIABLE #S 128 ALLOT
 8 : SCR 0 #S 2+ #S 2! ;
 9 ( Precedence ) 1 +LOAD 2 +LOAD 3 +LOAD
10 : I 93 WORD ; INNEDIATE
11 ARITHMETIC DEFINITIONS
12 ( BASIC ) 4 +LOAD 5 +LOAD 6 +LOAD 7 +LOAD
13 : ( 10 *( +) ; IMMEDIATE
14 : ; E n] . ; I PRECEDENCE ;
15 FORTH DEFINITIONS
      74
                                                                   75
 0 ( Branching - high level )
                                                                ( Variables)
 2 : JUMP R> a >R ;
 3 : SKIP 0= IF R> 4 + >R THEN :
 4 : (NEXT) ( to \ step \ variable address -- )
 5 20UP +1 ( add step to var 3
     ⇒R 20UP R> ∂ SWAP (tstvs)
 5
 7
    OK IF SWAP THEN -
8 OF IF 2DROP R> 2+ ELSE R> 0 THEN >R ;
9 : [NEXT] COMPILE (NEXT) ;
19
:1
12
13
14
15
     76
                                                                   77
 0 ( Statement numbers)
                                                                ( BASIC )
 1 : FIND ( n - a) 1 05 0 05 2+ 00
       OVER I & = IF 2DROP I 2+ 0 LEAVE THER 4 +LOOP
 2
                                                                : FOR
     IF 0 SWAP 45 2 2! 85 2 2+ 4 85 +! THEN ;
 3
 4
 5 : RESOLVE ( n) FIND DUP @ DUP 0 ABORT" duplicated"
 6 BESIN TOUP WHILE DUP 2 HERE ROT ! REPEAT
     HERE NEGATE SHAP 1 ;
 8 : CHAIN ( n - a) FIND DUP 2 OC IF 2 NEGATE
 9 ELSE DUP & HERE ROT ! THEN ;
10
11 : STATEMENT ( n) HERE 2- 0 >R -4 ALLOT RPN CFA EXECUTE
12 R) RESOLVE [COMPILE] START ;
13
14
                                                                : 1F
15
```

72

```
O VARIABLE ADDRESS O VARIABLE #(
:) -10 #(+) #( a O< ABORT* Unsatched )*; IMMEDIATE
:DEFER (an an - an) #(2+
  BEGIN 20VER SHAP DROP OVER < NOT
   WHILE 25WAP DROP CFA , REPEAT ;
: PRECEDENCE ( n - ) IN @ [COMPILE] ' >R IN !
   (BUILDS , R) , IMMEDIATE DOES> 20 DEFER ;
: RPN ( n) 0 1 DEFER 2DROP #( @ OR ABORT" Syntax";
: NOTHINS ;
: START ( - a n) 0 0( ! 0 ADDRESS ! ' NOTHING 0
```

- ARITHMETIC ; IMMEDIATE : ?ISNORE #(# IF 0 1 DEFER 2DROP R> DROP THEN ;
- : INTEGER (BUILDS 0 , INMEDIATE DOES) [COMPILE] LITERAL ADDRESS & IF & ADDRESS ! ELSE COMPILE & THEN ;
- : (ARRAY) (aa-ap) SMAP >R 7 DEFER R> ICONPILEI LITERAL ADDRESS 2 IF 0 ADDRESS ! ELSE ' a 7 at a + 25MAP THEN : : [+] (: a = a) SMAP 1= 28 + : : ARRAY (n -) (BUILDS 28 ALLOT INNEDIATE

```
DOES) * [+] (ARRAY) ;
```

```
: [3+] ( x y a - a) ROT ROT R 1- OVER a t R> + 2t + ;
: 2ARRAY ( y x - ) (BUILDS DUP , t 2t ALLOT IMMEDIATE
    DOES> ' [$+] (ARRAY) ;
```

```
: LET STATEMENT 1 ADDRESS ! ; INMEDIATE
```

73

- (COMPILE) LET ; INMEDIATE
- : [1] COMPILE 1 NERE ; : TO RPN DROP ' [1] 0 ; INNEDIATE : STEP RPN DROP ' HERE 0 ; INNEDIATE
- : NEIT STATEMENT 20ROP ' INEXT] 0 1 ADDRESS ! ; INHEDIATE
- : REM STATEMENT IN @ C/L / 1+ C/L # IN ! ; IMMEDIATE
- : DIN COMPILEI REN ; INWEDIATE
- : STOP STATEMENT COMPILE ;S ; INNEDIATE
- : END STATEMENT 2DROP (COMPILE) ; [COMPILE] FORTH ; INMEDIATE
- : (GOTO) (GET#) COMPILE JUMP CHAIN . ; : GOTO STATEMENT (GOTO) ; INMEDIATE
- STATEMENT LOGIC ; INMEDIATE
- : THEN RPW O COMPILE SKIP (GOTO) ; INMEDIATE

Michael Perry 1981

```
0 ( Charles Moore's BASIC compiler, Input and Output )
1 : ASK .* ? * QUERY ;
2 : PUT (GET#) SWAP ! ;
3 : (INPUT) COMPILE PUT ;
4 : (,) ( n) (.) 14 OVER - SPACES TYPE SPACE ;
5 : , ( n) ?IGNORE ' (,) 1 DEFER ; IMMEDIATE
6 : " [COMPILE] ." 2DROP ; IMMEDIATE
7 INPUT DEFINITIONS
8 : , 716NORE RPN 0 (INPUT) 1 ADDRESS ! ; INMEDIATE
10 ARITHMETIC DEFINITIONS
11 : PRINT STATEMENT COMPILE CR ' (,) 1 ; IMMEDIATE
12 : INPUT STATEMENT 2DROP COMPILE ASK ' (INPUT) 0 IMPUT
13 1 ADDRESS ! ; INMEDIATE
14
15
     80
0 I Dwyer, page 17, Program 11 SCR
I INTEGER J INTEGER K
3 : RUN START
4 10 PRINT " THIS IS A COMPUTER"
5 20 FOR K = 1 TO 4
5 30 PRINT " NOTHING CAN GO"
7 40 FOR J = 1 TO 3
8 50 PRINT " WRONG"
9 60 NEXT J
10 70 NEXT K
11 80 END
12
13 RUN
14
15
      82
 O [ basic: input/print ] SCR
1 INTEGER K
 2 INTEGER X
 3 INTEGER Y
 4
5 : RUN START
 5 10 INPUT X , Y
7 20 LET K = X # Y ## 3
8 40 PRINT X , Y , K
9 80 END
10 ;5
11
12
17
14
15
```

78

```
79
( Operators)
LOGIC DEFINITIONS
: <> [ n n - t ] = NOT ; 2 PRECEDENCE <>
: <= [ n n - t 1 > NOT ; 2 PRECEDENCE <=
: >= [ n n - t ] < NOT ; 2 PRECEDENCE >=
                         2 PRECEDENCE =
ARITHMETIC DEFINITIONS
: = ( a n) SWAP !; 1 PRECEDENCE =
: ## ( n n ~ n) 1 SWAP 1 DO OVER # LOOP #;
6 PRECEDENCE ABS
5 PRECEDENCE ##
4 PRECEDENCE # 4 PRECEDENCE / 4 PRECEDENCE #/
3 PRECEDENCE + 3 PRECEDENCE -
2 PRECEDENCE < 2 PRECEDENCE >
  81
E basic: array demo 3 SCR
INTEGER K
9 ARRAY COCRDINATE
: RUN START
10 FOR K = 1 TO 9
20 LET COORDINATE K = ( 10 - K ) $$ 3
40 PRINT COORDINATE K + 5
60 NEXT K
80 END
RUN
    1
     1
     1
               Richael Perry
               1446 Stannage Ave.
     1
               Berkeley, Calif. 94702
     1
               (415) 526-8696
     ŧ
     1
```

Michael Perry 1981

.

1

t

1

1

.

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 <u>Godel</u>, 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:

- : FANCY-NOUN
 - 4 CHOOSE (select random number 0-3)
 - CASE 0 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 FANCY-NOUN 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 plus that is called in the middle of the definition is the <u>original</u> +, not the one being 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 compiling 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 FORTH's protection against recursion. Here are two recent contributions from our readers:

A Recursion 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. III, 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 -> ACKERMANN (m,n) :ACKERMANN (m n -- ACK) [SMUDGE] SWAP DUP 0= IF DROP 1+ ELSE SWAP DUP 0= IF DROP 1 - 1 ACKERMANN ELSE OVER SWAP 1 - ACKERMANN SWAP 1 - SWAP ACKERMANN THEN THEN; SMUDGE

Be aware of typing 3 4 ACKERMANN

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 immediately executed word to re-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.

Editor's note:

The technique that is generally preferred was described by Joel Petersen in the original article. It defines MYSELF as

- : MYSELF
 - LATEST PFA CFA, ; IMMEDIATE

or, for some other versions such as poly-FORTH:

: 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 SMUDGE 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 TODAY!

FOR TH DIMENSIONS III/6

0

Þ

tř

57) 54

ж

5e ar

æ

;n Te

Sw

⁷53

0:1

C4

ah

140

аđ

_:**`**

tes

ാട

. m

one

out Inr

173

out

thai The

۰an

3556

and

yœ

forr

vers

8080 ASSEMBLER

John J. Cassedy 339 15th Street Oekland, 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 ; I 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 byte before NEXT and a jump to two bytes before NEXT. Literally, PSH1 means push one level (HL) and fall into NEXT. I 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.

I 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: $0 = CS PE 0 \le$ and NOT.

I have opted to retain the immediate instructions MVI and LVI as opposed to an immediate flag #.

The 1MI 2MI 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 48 30H 0 (FIGFORTH 8080 ASSEMBLER 1 81AUG17 JJC 80MAR04) 1 HEX VOCABULARY ASSEMBLER IMMEDIATE : 8 [#] DUP + DUP + ; 2 'ASSEMBLER CFA '; CODE 8 + ! (PATCH ; CODE IN NUCLEUS) 3 : CODE ?EXEC CREATE [COMPILE] ASSEMBLER !CSP ; IMMEDIATE 4 : C; CURRENT @ CONTEXT ! ?EXEC ?CSP SMUDGE ; IMMEDIATE 5 : LABEL ?EXEC 0 VARIABLE SMUDGE -2 ALLOT [COMPILE] ASSEMBLER 6 !CSP ; IMMEDIATE ASSEMBLER DEFINITIONS 7 4 CONSTANT H 5 CONSTANT L 7 CONSTANT A 6 CONSTANT PSW 8 2 CONSTANT H 5 CONSTANT E 0 CONSTANT B 1 CONSTANT C 9 6 CONSTANT M 6 CONSTANT SP ';S 0B + @ CONSTANT (NEXT) 10 : 1MI <builds c,="" does=""> C@ C, ;: 2MI <builds c,="" does=""> C@ + C, ; 11 : 3MI <builds c,="" does=""> C@ C, C, ; 12 : 4MI <builds c,="" does=""> C@ C, C, ; 13 : 5MI <builds c,="" does=""> C@ C, ; : PSH1 C3 C, (NEXT) 1 - , ; 14 : PSH2 C3 C, (NEXT) 2 - , ; : NEXT C3 C, (NEXT) , ; 15 ;S</builds></builds></builds></builds></builds>	
Screen4931H81MAR22JJC 80MAR04)10 (FIGFORTH 8080 ASSEMBLER 281MAR22JJC 80MAR04)1100 1MI NOP76 1MI HLTF3 1MI DIFB 1MI EI207 1MI RLC0F 1MI RRC17 1MI RAL1F 1MI RAR3E9 1MI PCHLF9 1MI SPHLE3 1MI XTHLEB 1MI XCHG427 1MI DAA2F 1MI CMA37 1MI STC3F 1MI CMC580 2MI ADD88 2MI ADC90 2MI SUB98 2MI SBB6A0 2MI ANAAS 2MI XRAB0 2MI ORAB8 2MI CMP709 3MI DADC1 3MI POPC5 3MI PUSH02 3MI STAX90B 3MI DCXC7 3MI RSTD3 4MI OUTDB 4MI IN10C6 4MI ADICE 4MI ACID6 4MI SUIDE 4MI SBI11E6 4MI ANIEE 4MI XRIF6 4MI ORIFE 4MI CPI1222 5MI SHLD2A 5MI LHLD32 5MI STA3A 5MI LDA13CD 5MI CALL;S1415	
Screen 50 32H 0 (FIGFORTH 8080 ASSEMBLER 3 81AUG17 JJC 80MAR04) 1 C9 1MI RET C3 5MI JMP C2 CONSTANT 0= D2 CONSTANI CS 2 E2 CONSTANT PE F2 CONSTANT 0 : NOT 8 + ; 3 : MOV 8* 40 + + C, ; : MVI 8* 6 + C, C, ; : LXI 8* 1+ C, , ; 4 : THEN HERE SWAP 1; : IF C, HERE 0, ; 5 : ELSE C3 IF SWAP THEN ; : BEGIN HERE ; 6 : UNTIL C, ,; : WHILE IF ; 7 : REPEAT SWAP C3 C, , THEN ; : 8 ;S 9 10 11 12 13 14 14	

```
Screen 51 33H
O ( EXAMPLES USING FORTH 8080 ASSEMBLER 1 81AUG17
                                                                                      JJC 80MAR12 )
    FORTH DEFINITIONS HEX
  1
    CODE CSWAP ( WORD-1--- SWAPS HI AND LOW BYTE OF WORD ON STACK )
H POP L A MOV H L MOV A H MOV PSH1 C;
 2
                            ( FROM-2 QTY-1--- CONVERTS LOWER CASE TO UPPER )
 Ц
    CODE LCFOLD
          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
 5
 6
 7
 8
 q
           REPEAT NEXT C;
10
    ;S
11
12
13
14
15
Screen
               52
                       348
    ( EXAMPLES USING FORTH 8080 ASSEMBLER 2 81AUG17 JJC 80MAR12 )
  0
    CODE CMOVE (FROM-3 TO-2 QTY-1--- SAME AS IN NUCLEUS)
COL MOV B H MOV B POP D POP XTHL
BEGIN B A MOV C ORA 0= 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)
  1
  2
  3
  Ц
  5
  6
           CL MOV BH MOV B POP XCHG
H ENP B DAD XCHG XTHL B DAD
  З
           BEGIN B A MOV C ORA O= NOT
WHILE H DCX M A MOV D DCX D STAX B DCX
  q
10
           REPEAT B POP NEXT C;
DVE (FROM-3 TO-2 Q(Y-1--- SMART MOVE, DOES NOT OVERLAY)
>R 2DUP R> ROT ROT -
IF -CMOVE ELSE CMOVE THEN ;
11
12 : MOVE
13
14
15 ;S
              53
Screen
 Screen 53 35H
0 ( EXAMPLES USING FORTH 8080 ASSEMBLER 3 81AUG17 JJC 80MAR12 )
 1 80 CONSTANT CMMD ( COMMAND BYTE )
2 F0 CONSTANT CMMDPORT ( COMMAND PO
3 F1 CONSTANT STATUSPORT ( STATUS P
                                       ( COMMAND PORT )
T ( STATUS PORT )
           EL DELAY ( --- DELAY CONSTANT IN DE, DON'T USE THE STACK )
BEGIN D DCX D A MOV E ORA O= UNTIL RET C;
 4 LABEL DELAY
 5
                                                                                ( BIT MASK-1--- )
 6
    CODE STATUS
           H POF CMMD A MVI CMMDPORT OUT
1234 I LXI DELAY CALL
 9
           BEGIN
              STATUSPORT IN L ANA O= NOT
10
           UNTIL NEXT C;
11
12 ;S
```

Sieve of Eratostenes in FORTH

Mitchell E. Timin Timin Engineering Co.

The enclosed version of Eratosthenes S :ve 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 MHZ Z-80 machine, as were the others in the BYTE magazine article.

The speed improvement is primarily due 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 = 35
   CR = 35

0 ( The Sieve of Eratosthenes, after J. Gilbreath, BYTE 9/81 )

1 B190 CONSTANT SIZE SIZE STRING FLAGS ( make array of flaos )

2 : PRIME 0 FLAGS SIZE 1 FILL ( start by setting the flags)

3 0 ( create counter which remains on top of stack )

4 SIZE 0 D0 ( repeat following loop B190 times )

5 I FLAGS C0 ( fetch next flag to top of stack )

6 IF ( if flag is true then do the following )

7 I DUP = 3 e ( calculate the prime number )

8 DUP I + ( stack is counter, prime, K )

9 BEGIN DUP SIZE < WHILE ( repeat for K < B190 )

10 0 OVER FLAGS C! ( clear Kth flag )

11 OVER + ( stack prime to K )
  10
                                            OVER
                                                                                                     ( add prime to K )
   11
                                    REPEAT
   12
   13
                                    DROP
                                                DROP 1+ ( drop K & prime, increment counter)
  14
15
                          ENDIF
                 LOOP
                                 3 SPACES
                                                       . .* PRIMES * ; ( finish: display count)
SCR = 36
   0 ( 1 1
            testing the sieve algorithm )
BELL 7 EMIT 1
                                                                                        O VARIABLE KOUNT
    2 : NEW-LINE CR O OUT ! :
3 : NEW-LINE? OUT 0 70 > IF NEW-LINE ENDIF
   2 :
                                                                                                                    .
    5 1 PRIME-TEST
    5 1 PRIME-TEST BELL (first sound the bell)
6 10 0 DO PRIME LOOP BELL (run the prime finder 10 X)
7 (above is for timing test, below is for validation)
                                                                             ( clear counter, start new line )
( check each flag )
( see if it's set )
( calculate the prime number )
                 O KOUNT !
SIZE O DO
I FLAGS
                                       I NEW-LINE
                                            CĐ
  10
                          IF I DUP + 3 +
7 + R NEH-LIME?
  11 12
                                                                                  display it
  13
14
15
                                       KOUNT +1
                                                                              ( count it )
                          ENDIF
                                 CR KOUNT ? . PRIMES
                                                                                                  >> ( dimplay the count)
                 1.000
```

SKEWED SECTORS FOR CP/M

Roger D. Knapp

;

:

:

i

In regard to Michael Burton's article in FORTH DIMENSIONS, III/2, page 53, "Increasing fig-FORTH Disk Access Speed," I enclose a simple mod to the 8080 or Z80 assembly list to effect the CP/M skewed sector disk I/O. The FORTH routines I used to test the scheme are included. The first cluster or screen is offset by 52 sectors so that the operating system is transparent and acreens 0 and 1 hold the directory. I move the message acreens to SCR# 24 and 25 leaving 2-20 for the FORTH binary program run by CP/M or CDOS.

In order to check any increase in disk access speed I timed the following operation with a 10 screen buffer:

20 270 10 MCOPY 20 270 10 MCOPY 20 270 10 MCOPY

Elapsed times were 204 and 138 seconds for straight and skewed sectors respectively. Note that this reflects disk access speed for read/write of several sequential sectors and in no way compensates for inadequate planning or poor programming in other disk I/O applications.

If this seems trivial, then you have no need for CP/M file compatible I/O. My motive for these changes is the desire to write the assembler program for fig-FORTH via modem (easy to implement in FORTH) to friends and colleagues. As added value my disk I/O can be faster.

201 DE, SETDSK : SEND DRIVE + TO CP/M LD 105 CALL ; RESTORE (IP) POP BC NEXT JP 0,1,7,13,19,25,5,11,17,23,3,9 TATAL: DB DB 15.21.2.8.14.20.26.6.12.18.24.4.10.16.22 S-SKE' DB 53 ADDED '\!'+80H na AFTER SETORV-12 DW SSKEW: DW \$+2 "SET DRIVE" ; SECTOR SEQUENTIAL POP DΕ ; TRANSLATION TABLE ABOVE LD HL, TRTBL ; ADDR OF NEW SECTOR ADD HL DE 10 E,(HL) PUSH DÉ ; SECTOR TRANSLATED .1P NEXT ÷10т : TASCALC DB 87H 'T&SCAL' 0B 28 1C1+80H SSKEW-9 MODIFIED DW) DOCOL, DENSTY TSCALC: DW D٧ AT ZBRAN, TSCALS-S D'A' DOUBLE DENSITY DW LIT, BUPDR2 *20T SINGLE DENSITY SKID 52 SECTORS FOR TSCALS: DH LIT,52,PLUS LIT, EUPDRI DW OPERATING SYSTEM DW SLHOD LIT,MXDRV-1 กบ D¥ MIN DUP, DRIVE 0N DW AT , ÉQUAL ZBRAH, TSCAL3-S nu DV: DSUD BRAN, TSCAL4-S 22 TSCAL3: DV DRIVE, STORE <u>D:/</u> SETDRV LIT, SEPTRI TSCALA: DX 21 SLMCD, TRACK DW STORE, ONEP SSKEU SEC, STORE 0K : SEQUENTIAL TO CP/M SKEW 0% D'4 SEMIS SCR # 5/2 (CPM style cisk layout and 1/0) 1 FORTH DEFINITIONS DECISAL 3 LASEL 105 (CP/H SERVICE REQUEST) 1 LDHLM, D ADDP, JPHL, G: ٨ 5 CODE SET-10 (sector track addrs ---) A H POP, B PUSH, H B LD, L C LD, 21 D LDP1, IOS CALL, B POP, H POP, B PUSH, H B LD, L C LD, 18 D LDPI, 105 CALL, B POP, H POP, B PUSH, L C LD, 1E D LDPI, 105 CALL, B POP, 7 H POP, B PUSH, Q Ó NEXT, C: 10 11 CODE SET-DRIVE (n ---) L C LD, 19 D LDPI, IOS CALL, B POP, H POP, B PUSH, 12 NEXT, C; 13 14 15

Timothy Huang 9529 NE Gertz Circle Portland, OR 97211

While I was in the process of explaining the disking to some friends, I found it would be nice to show them some sort of representation which lists all the disk buffer status. This short program was then written for this purpose.

The figFORTH uses the memory above USER area for the disk buffer. This disk buffer area 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 6080 CP/M, that set it to be 128 bytes. Another constant beside B/BUF frequently referred in disking is the B/SCR (buffers per screen). For B/BUF = 1024, the B/SCR = 1 and for B/BUF = 128, B/SCR = 8.

Each block needs 2 bytes in front of it as the header which contains the update bit (bit 15) and block number (lower 0-14 bits). It also needs a 2-byte tail to end the block.

The word BLOCK will put the beginning address 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 exhausted or you terminate it by pressing any key. 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 column. 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 handier to know which subpart of a given acreen the block I want.

Lines 12 and 13 check the early termination and finish the definition.

) O (SECTOR SKEW FOR CP/H FORMAT CLUSTERS 1 FORTH DEFINITIONS DECIMAL 2 : CTABLE (bytesize TABLE) 3 <8UILDS 0 D0 C, LOOP D0ES> + C0 ; 4 22 16 10 4 24 18 12 6 26 20 14 R 2 21 15 9 3 23 17 11 5 25 19 5 13 7 1 0 27 CTABLE S-SKEW (for CP/M clusters) : MSETUP (Setup n sectors for NXTS.) 7 (adrs blk n --- sec trk addr ... secn trkn addrn) POT DVER 128 * + ROT ROT OVER + 1- SWAP 1- SWAP 8 0 DO I 26 /MOD SWAP I+ S-SKEW SWAP ROT 128 - DUP 10 -1 +LCOP DROP : 11 12 13 : MRTS (Read n sectors.) (s t a ... sn tn an n ---) 10 0 DO SET-10 SEC-READ DISK-ERROR @ IF LEAVE THEN LOOP ; 15 SCR # 62 ۱ O (MORE CP/M FORMAT DISK 1/0 1 FORTH DEFINITIONS DECIMAL 2 3 ٤ 5 : R/M-CP/M (CP/M skewed cluster I/0.) 7 (addrs blk f ---) >R 52 + 2000 /MOD SET-DRIVE SEC/HLK MSETUP (52 + sn cluster alloc CP/M) 0 SEC/BLK NRTS e R> 1F 10 ELSE SEC/PLK MUTS ENDIF DISK-ERROR # 8 PERROR ; 11 12 13 / 411 of screens 61 and 62 shamelessly adapted from John James') 14 (fic-FORTH for the LSI-11. 15 04

SCR # 90

SCR # 51

7/11/81 O (.BUMS TDH) 1 DECIMAL : .BUFS (display adr of all buffers) 2 ,BUFS (display adr of all bullers / CR , # # Addr(hex) Upd Block# Soreen -ST FIRST #BUFF 1+ 1 DO CR I 2 ,R 2 SPACES DUP 2+ HEX 6 0 SWAP D.R DECIMAL 3 SPACES DUP ● 32768 AND -sub* ú 56 0= 0= 32 + EMIT 2 SPACES DUP @ 32767 AND DUP 6 .R 4 SPACES B/SCR /MOD 5 .R 4 SPACES 2 .R 132 + ?TERMINAL IP LEAVE THEN 8 Q 10 11 LOOP DROP CR ; 12 13 14 15 OK BUES Addr(hex) Upd Block# Screen -sub # 1 3E82 720 90 0 1 2 2 3P06 721 90 722 723 3**F**8A 90 3 34 ú 400E 90 90 90 56 4092 724 4116 725 567 **90** 419A 726 8 421E **9**0 727 Ó 4282 0 Q 0

Ē

FLOATING POINT ON THE TRS-80

Kalman Fejes Kalth Microsystems PO Box 5457, Station F Ottawa, Ontario K2C 331 Canada

BLOCK 9

F#

Fa

F I

floating point number on stack in

memory at location A.

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 precision 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

- F + (F1F1 - F)Add (F = F2 + F1)F -(F2F1--F) Subtract (F=F1-F1) F * (F2F1 -- F) Multiply (F = F2 + F1)
- F/ (E|E| - E)Divide (F=F2/F1)

0 (FTP #1 :KIF 810816) FORGET FTASK : FTASK ; HEX 1 (SINGLE PREC. FLOATING POINT FOR TRS-80 IN MMSFORTH V1.8) 2 : EXX D9 C, ; 3 CODE F.& EXX OFBD CALL 28A7 CALL EXX NEXT 4 CODE F#& OESC CALL EXX HL POP 2 RST OAB1 CALL EXX NEXT DUP 2 + @ SWAP @ 4 40AF C! ; DUP ROT SWAP ! 2 + ! 4 40AF C! ; 5 6 : F0 : FI 7 8 : A S 4121 PO 9 : F#0 10 : F#IN HERE O OVER 3E FILL BL WORD F#& A S; "? PAD DUP 1+ 63 EXPECT F#& A S; F#O SWAP (L) (L) , (L) (L) , ; P#0 SWAP (L) (L) , (L) (L) , STATE CO IF F#1 BLSE P#0 THEN ; S A F.& 4 40AF C: ; 11 : F#1 12 : F# IMMEDIATE 13 : F. 14 : 10FT ; DECIMAL 15 BLOCK 10 0 (FLOT. PT. #2 : IF 810816) FORGET 10FT : 10FT : 1 HEX 2 CODE F+& EXX DE POP BC POP 716 CALL EXX NEXT 3 CODE F-& 4 CODE F*& 713 CALL 847 CALL EXX DE POP BC POP EXX NEXT EXX DE POP BC POP EXX NEXT CODE 2/& 5 EXX DE POP BC POP 8A2 CALL EXX NEXT 6 : F+ : F* SA P+& AS : F-: F/ S A P_& S AS; AS; A 7 F*& SA S A F/& AS 8 DECIMAL 9 (SAMPLE AND TEST ROUTINES) F#IN CR F# 2 F+ F# 200.0E-2 F-F# 5000.1 F* F# 5.0001E+3 F/ 10 : FTEST 11 12 PAD F! PAD PO F. : 13 14 ;S 15 -- F) F TEST (---) Takes a number from the current A sample program to demonbuffer, converts it to single prestrate the use of these floating cision floating point number and point operators. It asks for a floating point number from the leaves it on the stack. keyboard, manipulates it using all F#IN (-- F) the operators defined and prints Asks for a floating pint number the result. (It should be the same from the keyboard, and leaves it number that was supplied.) on the stack. Notes: A -- 16 bit address (A -- F) Floating point fetch. Takes a F, F1, F2 -- are single precision floating point number from floating pint numbers (two 16-bit memory at address and leaves it words each). on the stack. (F A --) Floating point store. Stores the

TURNING THE STACK INTO LOCAL VARIABLES

Marc Perkel Perkel Software Systems 1636 N. Sherman Springfield, MO 65803

SCR # C

С

D

E

F

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 return the volume, surface area, and length of edges. Try it!

For this kind of siluation I 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, with S1, S2 and S3 returning the top 3 stack values that were there before 3 ARGUMENTS was executed. S4 through S9 are zero-filled and the stackpointer is set to just below S9.

S1 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 S4 writes a 5 into S4. 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 was 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 }***) 0 VARIABLE CARGI VARIABLE [TO] 1 : HARG CREATE , DOES> @ LARGI @ SWAP - LTOI @ ?DUP 2 IF O< IF +! ELSE ! ENDIF ELSE @ ENDIF O [TO] ! ; 3 4 4 +ARG S3 C +ARG S7 2 +ARG S2 6 +ARG S4 5 0 +ARG S1 E +ARG S8 6 8 +ARG 55 A +ARG S6 (*TO VARIABLES*) 10 +ARG S7 7 8 (*SETS STORE FLAG FOR +ARG*) 9 ; TO 1 (TO] ! ; (*SETS +STORE FLAG FOR +ARG*) Α : +TO -1 [TO] ! ; Ĥ : ARGUMENTS R> CARGI @ >R >R 2# SP@ + DUP CARGI ! 12 - SP@ SWAP С - 2/ 0 DO 0 LOOP 0 ET03 ! ; Ð : RESULTS 2* LARGI @ SWAP - SP@ - 2/ E O DO DROP LOOP R> R> LARGJ ! >R ; 2

0 (ARGUMENT EXAMPLE --- BOX COMES IN WITH HEIGHT, LENGTH & WIDTH AND LEAVES VOLUME, SURFACE AREA & LENGTH OF EDGES) 3 : BOX **3 ARGUMENTS** (VOLM) S1 S2 S3 * * TO S4 S1 S2 S3 * * S2 S3 2 * * S1 S3 2 * * + + TO S1 4 5 (SURE) S1 4 * S2 4 * S3 4 * + + TO S3 (EDGE) 6 7 S5 T0 S2 3 54 TO 51 3 RESULTS # 9 A ₿

SCR # 20 : TASK ; 0 : DISKO! 5 ARGUMENTS 1 S1 S2 0400 U/MOD 1+ TO S1 TO S2 2 BEGIN S4 0> 3 WHILE SI BLUCK 52 + 53 4 S5 IF SWAP UPDATE ENDIF 5 S4 0400 S2 - MIN DUP TO S6 CHOVE 6 7 S6 +T0 S3 56 NEGATE +TO S4 8 9 1 +TO S1 0 TO S2 A REPEAT R С 0 RESULTS # : DISK@ O DISK@! ; Ď Ε : DISK! 1 DISK@! # F -->

ł

T

1

t

ł

F

e

P 1

٩

¥

t

t

FORTH DIMENSIONS III/6

GRAPHIC GRAPHICS

California College of Arts and Crafts

Accompanying these comments are several 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.

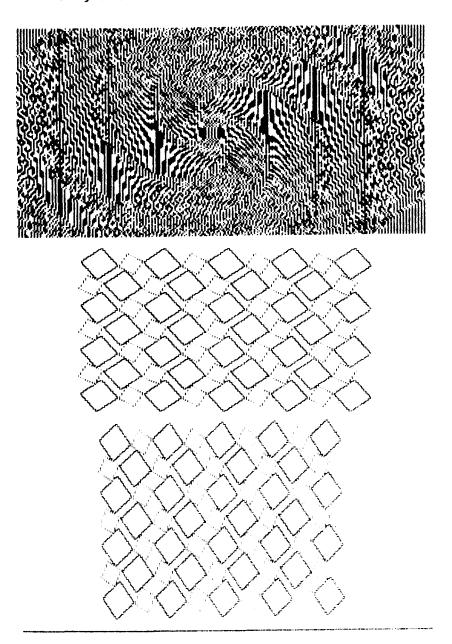
While these graphic samples could have been programmed in other languages, I have found the advantages of using FORTH are both practical and expressive: immediate and modular experimentation with the peculiarities and limitations of the Apple video display, and orchestration of complex visual effects 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.

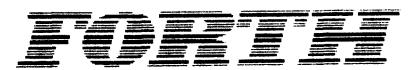
÷

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 help 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 II using Cap'n Software FORTH, as cover artwork for their book "Some Common BASIC Programs":





CASES CONTINUED

Editor's Note: In Volume II, Number 3, FORTH DIMENSIONS published the results of FIG's CASE Statement Contest. As we had hoped, the variety of responses has stimulated further work on the subject. Here are four additional CASE constructes submitted by our readers.

Eaker's CASE for 9080

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 useful.

O (CASE STATEMENT BY CHARLES EAKER FD II 3 39 J 1 : CASE ?COMP CSP @ ICSP 4 ; IMMEDIATE 2 CODE (OF) H POP D POP ' - 8 + CALL L A MOV H ORA O= 3 IF B INX B INX NEXT ENDIF D PUSH ' BRANCH JMP C; JJC 81AUG09) : OF 4 ?PAIRS COMPILE (OF) HERE 0 , 5 ; IMMEDIATE à 5 : ENDOF 5 ?PAIRS COMPILE BRANCH HERE O 6 SWAP 2 [COMPILE] THEN 4 ; IMMEDIATE , : ENDCASE 4 ?PAIRS COMPILE DROP 7 BEGIN SPE CSP E = O= WHILE 2 [COMPILE] THEN 8 q REPEAT CSP ! ; IMMEDIATE TEST CASE 41 OF ." A " ENDOF 42 OF ." B " ENDOF 65 OF ." e " ENDOF 10 11 : TEST CASE 12 ENDCASE : 13 14 (41 TEST A OK) 15

Eaker's CASE Augmented

Alfred J. Monroe 3769 Grandview Blvd. Los Angeles, CA 90066

I was delighted with Dr. Esker's CASE construction (FORTH DIMEN-SIONS, Vol. II, No. 3, p. 37) and implemented it immediately. Recently I have found it desirable to augment CASE with three additional constructs in order to treat ranges of variables. It has occurred to me that 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 is 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, $\langle OF, \rangle OF$, and RNG-OF. $\langle OF does a$ "less than" test. $\rangle OF does a$ "greater than" test. RNG-OF does an inclusive range test. $\langle OF and \rangle OF$ are trivial modifications of OF and $\langle OF \rangle$. RANGE and RNG-OF are constructed in the same spirit as $\langle OF \rangle$ and OF.

Screen 144 compiles to 175 bytes. Screen 145 compiles to 223 bytes. SCR # 144 & (DR. EHKER'S CHSE CONSTRUCT WITH H SLIGHT MUDIFICATION) 1 : CHSE YOUMP CSP & YOSP 4 3 1MMEDIATE 2 : COP OUER = 14 DROP 1 ELSE & ENDIF 3 I UP 4 YPAIRS COMPILE (OF) COMPILE OBRANCH 3 HERE 0 , 5 3 IMMEDIALE 4 : ENDUR 5 YMHIRS COMPILE BRANCH HERE 6 . SWAP 2 5 LCOMPILES ENDIE 4 3 IMMEDIATE 6 : ENDCHSE 4 YPHIRS COMPILE DRUP BEGIN SPO CSP 0 = 0 = 7 WHILE 2 LOUMPILES ENDIF REPEAT CSP ! & IMPEDIATE 8 9 10 11 12 13 14 15 ---> SCR # 145 0 C THE CUE, OUE, HND RNG-OF EXTENSIONS > 1 : (KOF) OUER > 1F DRUP 1 ELSE @ ENDIF 3 * KOF 4 VEHIRS CONFILE (KOF) COMPILE GERANCH 2 3 HERE 0 5 5 11MHED1HIE 4 : COUPD DUER < 1F DROP 1 ELSE 0 ENDIF 3 5 + JUH 4 PHARS CUMPILE (JOF) COMPILE BERFINCH HERE 0 , 5 ; IMMEDIATE 6 : RHNUE > UUER DUP R> 1+ < 1F SMRP 1- > 1F DROP 1 ELSE & ENDIF ELSE DROP DROP 0 ENDIF J 8 9 : HNG-UN 4 YPHINS COMPILE RANGE COMPILE BERANCH HERE 8 , 5 ; 10 IMMEDIATE 11 12 13 14 15 -->

```
Screen 147 illustrates a pre-Esker
SCR # 146
                                                                                  solution to the design of an interactive
  6 ( EXAMPLE USE OF AUGMENTED CASE )
                                                                                  terminal input that places a hexadecimal
  1 48 CONSTANT "0" 57 CONSTANT "9" 65 CONSTANT "A"
2 70 CONSTANT "F" 13 CONSTANT "CR"
                                                                                  number on the stack, and which provides
                                                                                  for error detection and error recovery. It
  3 3 CONSTANT ONTRE-C
                                                                                  is. of course written in my usual sloppy,
                                                                                  unennotated, semi-readable fashion.
  5 & UHRIABLE FLAG
  б
                                                                                     Screen 148 offers a neater solution in
  V : SYN-ERR OR . " SYNTAX ERROR, REENTER NUMBER " OR
                                                                                  terms of <OF and >OF. It is definitely
  S URUP DIKUP G "G" 3
9 I C-HBUKI CR ." CUMMAND ABORT " CR DROP DROP GUIT 3
                                                                                  more readable. Screen 149 offers a still
                                                                                  neater solution in terms of RNG-OF.
 16
 11
                                                                                  Screen 147 compiles to 160 bytes, screen 148 to 176 bytes, and screen 149 to
 12 : VABORT ONTRL-C = IF DROP CR . " COMMAND ABORT " CR GUIT
                            ELSE DUP ENDIF #
 13
                                                                                  144 bytes. Need I say more?
 14
  SER # 147
   6 C GET-HEX LEAVE A HEX # ON TOP OF STACK >
   1 C A FRE DR. EAKER SOLUTION TO AN INTERACTIVE TERMINAL INPUT >
   4 : GET-HEX 0 FLAG ! 0 BEGIN KEY DUP DUP EMIT ? REORT
                 13 = IF 1 FLAG ! DROP
   5
                     ELSE DUP "6" < 1F SYN-ERR
ELSE DUP "9" > 1F DUP "A" < 1F SYN-ERR
ELSE DUP "F" > 1F SYN-ERR END1F
   6
   1
                                                                                   SEND & CHECK TO FIG TODAYI
   ŝ
                     ENDIE ENDIE ENDIE ENDIE
                                                                                    MAKE THIS YOUR BEGINNING!
     FLHG @ 0= 1F 48 - DUP 9 > 1F 7 - ENDIF SHIP 16 + + ENDIF
  111
                                                                                              RENEW NOW!
  11
  12
        FLAG & UNTIL :
  13
  14
  15
 SCR # 148
   WICH NEATER SOLUTION TO THE TERMINAL INPUT ROUTINE >
   1 : GET-HEX & FLAG !
                                                                                                      FOR,
          U BEGIN KEY DUP DUP EMIT
   2
   5
                   CASE CNIRL-C OF C-ABORT
                                                       ENDOF
                           "CR" OF I FLAG ! DROP ENDOF
"G" (UF SYN-ERK ENDOF
"F" >OF SYN-ERK ENDOF
   4
   5
   6
                     "9" 1 +
   2
                                 <0F 48 -
                                                       ENDOF
                     "A" 1 -
                                       ×0F 55 -
   з
                                                       ENDOF
                                 SYN-ERR
   - <del>'</del> 4
  10
                 ENDCASE
  11 FLHS & G= 1F SWAF 16 * + ENDIF
12 FLHS & UNTIL ;
  13
  14
 15
SCR # 149
   # K H STILL NEATER SOLUTION >
                                                                                              RENEW TODAY!
   1 : GEI-HEX & FLHG !
   2. 6 BEGIN KEY DUP DUP EMIT
     CASE UNTRE-C OF C-ABORT
   тъ.,
                                              ENDOF
            "CR" OF 1 FLAG ! DROP
"0" "9" RNG-OF 48 -
                                              ENDOF
   5
                                              ENDOF
            "A" "F" RNG-OF 55 -
   6
                                              ENDOF
              SYN-ERK
   12
   8 ENDORSE
   9 FLAG @ 0= IF SWAP 16 * + ENDIF
  16 FLAG @ UNTIL ;
  11
  12
  13
  14
  15
```

CASE as a Defining Word

After reading the CASE contest 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 <u>very simple</u>. There is no error checking since the form is so simple the most novice prodrammer can use it.

CASE is analogous to vectored GOTO in other languages. Its usage with my words is:

CASE		NAME	
Α	IS	FUNCTION A	
в	IS	FUNCTION B	
С	IS	FUNCTION C	
(etc.)			
OTHERS ERROR FUNCTION			

General usage would be as a menu selector; for example, you print a menu:

1	BREAKFAST
2	LUNCH
3	DINNER

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:

15

CASE		MEAL
1	IS	BREAKFAST
2	IS	LUNCH
3	IS	DINNER
OTI	HERS	NO MEAL

You have previously defined BREAK-FAST, LUNCH, DINNER and NO MEAL.

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> pertion 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.

106 0 (CASE NAME IS FUNCTION-A PAIR = VALUE-A 1 A ADDR OF FUNCTION-A FUNCTION-B 2 в IS 3 С IS FUNCTION-D ETC. 4 ERRORFUNCTION 5 OTHERS) 6 : CASE CREATE HERE 0. , (AT COMPILATION BUILDS HEADER, LINK 7 POINTS TO ADDR OF + OF PAIRS HERE SET TO ADDR OF VALUE-1) 8 9 (AT EXECUTION, ADDR OF #OF PAIRS) 10 DOES> 1 ROT ROT DUP 2+ SWAP @ 0 DO 2DUP @ = IF DUP 2+ @ 11 (COMPAIRS INPUT VALUE 12 EXECUTE ROT DROP O ROT ROT (WITH VALUE A, B, C, ETC, AND) 13 LEAVE ELSE 2+ 2+ THEN LOOP (EXECUTES ASSOCIATED FUNCTION) 14 ROT IF @ EXECUTE ELSE DROP THEN DROP ; 15 107 O (CASE WORDS) 1 : IS , ', 1+ ; (HERE, PAIR -- HERE, NEXT-PAIR) 2 : OTHERS ', SWAP I ; (HERE, &-OF-PAIRS) 3 4 5 6 7 8 9 10 11 12 13 14

THIS IS THE END! THE END OF VOLUME III THE END OF YOUR MEMBERSHIP? DON'T LET IT HAPPEN! RENEW TODAY!

Generalized CASE Structure in FORTH

E.H. Fey

Introduction

The CASE CONTEST held by FIG last ended with some excellent vear 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 fig-FORTH 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 Glossary 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 emminently 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.

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 ;

where

: <LIST <BUILDS SMUDGE CSP] ; : IX (k pfa...adr) SWAP 1 MAX 1 - DUP + +;

and is used in the form:

CASE: xxxx cfal cfa2 cfan ;

to define a case selector word named XXXX.

When the new word, xxx , is executed in the form

(k=1,2,...,n) k xxxx

the k'th word in the list will be executed. For example, define the following words, COW, CHICK, PIG, and BARN:

: COW ." MooOOoo"; : CHICK ." Peep"; : PIG ." Oink" ; CASE: BARN COW PIG CHICK ;

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 xxxx. 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 can enter the list. The list is terminated by ; which completes the definition of xxxx .

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 ? resumed the compilation state.

The point is that <LIST is a powerful word for entering named lists and data 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.

The Multi-Purpose Case 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 xxx . At execution time, the number is retrieved and used to select the appropriate DOES> part for the case type of xxxx . The type number is transparent to the user.

The definition of the new case compiler is:

- : MCASE: <BUILDS SMUDGE :CSP HERE 1 C, 0 C,] DOES> DUP C@ DOESPART;

where DOESPART is a case selector word defined by CASE: .

The <BUILDS part of MCASE: compiles a "1" 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 xxxx , 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.

3 OVER C!; IMMEDIATE 5 OVER C!; IMMEDIATE : KEYED : RANGE

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

: DEFAULT: ?COMP EOL , HERE OVER CO [DEF] ; IMMEDIATE

where EOL is an end of list terminator constant defined by

' IS CFA CONSTANT EOL

and [DEF] is a case selector word defined by CASE: .

165

165

165

165

165

165

165

165

165

166

166

165

165

165

166

166

167

167

167

167

167

167

167

167

167

169

168

168

163

168

168

169

167

167

169

169

169

169

169

167

DEFAULT: first checks to see that you are in the compile state since you should be compiling xxxx . It then enters the end of list terminator, EOL , to the dictionary. Finally it takes the parameter field address of xxxx left on the stack by the <BUILDS part of MCASE: , gets the type of xxxx and executes the case selector word [DEF] depending on the type of xxxx 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 xxxx . 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.

Execution of Case Selector

All case selector words, xxxx , defined by MCASE: are executed in the form:

k xxxx

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 available. Let's consider first the lists with single word execution elements.

Single Word Execution Elements

(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,...,n . Otherwise the default case, cfad, will be selected and executed.

(2) Keyed type

N

ACASE: xxxx	KEYED
[kl ,] cfal
[k2 ,] cfa2
[kn , DEFAUL] cfan _T: cfad ;

When xxxx is later executed in the form k xxxx , the case cfai will be executed if a value of k=ki is found in the list. Otherwise, the default case, cfad, will be executed.

(GENERAL CASE STRUCTURE EHF 10/23/81 165 0) 1 165 2 (EXECUTION VARIABLES AND ARRAYS als Kim Harris, Bute Aug '80) 3 (PP 184 also see M. A. McCourt: FD II/4 PP 109. EHF 2/11/81) 4 5 : IX (K pfa...adr) (Computes adr of index K = 1, 2, ..., n) 165 6 (... Pfa kmax1) 7 SHAP 1 MAX 8 1 - DUP + + # (... pfa+2[k-1]) 9 : <LIST (General <BUILDS word to construct named lists) 165 10 (BUILDS SMUDGE !CSP] ; 145 11 165 12 ' : CFA @ CONSTANT COLON (For headerless code definitions) 165 13 S CFA CONSTANT EOL (End of list delimiter) 165 14 165 15 -+> : CASE: KLIST DOES> 1x @ EXECUTE # 166 0 1 (Used in the form: CASE: XXXX cfal cfal...cfan ;) (to create an execution array XXXX with initial values cfal.) 2 (cfa2,...cfan which are code field addresses of previously) ٦ (defined words. Executing XXXX in the form: - K XXXX 4 (will produce the execution of cfak + k= 1+2+...+n 5) 6 : LIST: <LIST DOES> IX # # 7 (Used in form: LIST: XXXX E n1 + n2 + n3 + +] + 8 (to create a list of constants named XXXX . Executing XXXX) 9 (in the form: k xxxx will leave nk on the stack. 166 10 166 11 : XEQVAR: <LIST DOES> @ EXECUTE # 166 12 (Used in the fore: XEQVAR: xxxx cfa; 166 13 165 14 (to create an execution variable xxxx with an initial value) 166 15 (of which is an existing word. Executing xxxx causes) --> (of a to be executed. The word of a way be chansed by usins) ٥ 1 (INSTALL mono AT XXXXX where mono is the new word.) 2 : INSTALL (... of a) COMPILED ' STATE @ IF COMPILE CFA ELSE CFA 3 THMETITATE 4 THEN : 5 : AT (cfa...) [CONFILE] ' STATE @ IF COMFILE 24 COMPILE ! 6 INMEDIATE 7 ELSE 2+ ! THEN # 8 : (ATKIN) (k cfa pfa...) ROT 1 MAX 2 # + ! ; (Stores cfa at) 9 167 10 (adr#2k+pfa where K#1:2:...:n Compiled by ATKIN .) 167 11 167 12 (K cfa...) [COMPILE] ' STATE & IF COMPILE (ATKIN) : ATKIN 167 13 ELSE (ATKIN) THEN ; INMEDIATE 167 14 (Used in form: K INSTALL of a ATKIN XXXX 167 15 (where xxxx is an execution array defined by CASE: + cfa) ----(is the new word to be installed as element k=1,2,...,n ۵ 3 1 : DUN ; 168 2 --> 3 168 (NOTE: NeCourt's implementation of the function INSTALL ATKIN, 4 5 does not work inside • : definition. The above does. 6 168 7 MCASE: . A GENERALIZED EXTENSION OF CASE: 8 168 9 1. Three types of case stuctures: 168 10 a. POSITIONAL (default) 168 11 b. KEYED 168 12 C. RANGE 2. Two structure options for each type: a. SINGLE WORD EXECUTION (default) 168 13 168 14 b. HIGH LEVEL HEADERLESS CODE SEQUENCE 169 15 ٥ (Define BOESPART and IDEF] as Execution arrays to be filled) 1 167 2 (in later) 3 CASE: DEESPART DUN DUN DUN DUN DUN DUN J (6 Cases) 4 5 CASE: [DEF] DUN DUN DUN DUN DUN DUN DUN F 6 7 : NCASE: (The seneralized case compiler) 169 8 (BUILDS SNUDGE !CSP HERE (Leave pfa on stack)

	AW/ AV
(3) Range type	169 11
	167 12
MCASE: XXXX RANGE	169 13
[L1, Hi,] cfal	169 14
[L2, H2,] cfm2	169 15
	170 0
[Ln, Hn,] cfen	170 1
DEFAULT: cfad :	170 2
	170 3

140 14

176 4 170 5

170 6

170 7

170 9

176 10 170 11

170 12

170 13

170 14

170 15

171 0

3

6

9

171 1

171 2

171

171 4

171 5

171

171 7

171 8

171

171 10

171 11

171 12 171 13

171 14

171 15

172 0

3 172

4 5 172

172 1 172 2

172

172 6

172 7

172 8

172 9

172 10

172 11

172 12

172 13

172 14 172 15

173 0

173 1

173 3 173 4

173 5

173 6

173 7

173 8

173 9

173 10

173 11

173 12 173 13

173 14

173 15

174 4 174 1

174 2

174 5

174 3 174 4

173

170 8

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 Hi , followed by the execution element, ofai.

When xxxx is later executed in the form k xxxx , the case cfai will be selected if the condition

 $Li \leq k \leq Hi$

is found during a search of the list. If not, the default case, cfad , will be executed.

Headerless Code Execution Elements

Instead of specifying the execution elements as previously defined FORTH words, the elements may be specified as a sequence of FORTH words in the form:

H: ;H

or as

DEFAULT:seq ;

whereseq 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

MCASE: xxxx H:seq1... ;H H:seq2... ;H H:seqn... ;H DEFAULT:seqd ;

(2) Keyed type

MCASE: XXXX KEYED [k1 ,] H: ...seq1... ;H [k2 ,] H: ...seq2... ;H [kn ,] H: ...seqn... ;H DEFAULT;seqd.... ;

1 C+ (Default type = 1) 0 C, (Number of cases in list = 0) (Enter compile state for list) 1 (Gets type) DOES> DUP CO (Executes appropriate search) DOESPART : : DEFAULT: (pfs...) (Mandatory word used after caselist in) (an MCASE: definition. Compiles (S ,) TCOMP EOL . HERE OVER CO (... Ffs adth type) [DEF] ↓ IMMEDIATE : KEYED (pfa...pfa) (Used after MCASE: xxxx to set casetype=3) 3 OVER CI + IMMEDIATE : RANGE (Pfs ... Pfs) (Used after MCASE: xxxx to set type=5) 5 OVER CI J INMEDIATE : N? (n pfa...n pfa f) (Checks for valid casecount, n , with) (count in case list with pfa specified. True if valid.) OVER OVER 1+ CP (... B Pfa n count) (... n Pfa n count) OVER 1 < >R > R > OR 0 = i--> (POSITIONAL TYPE WITH SINGLE WORD EXECUTION OPTION, TYPE 1) : PSFIND (n pfa...) (Type 1 case for DOESPART, finds and) (executes case n or default if n<1 or n>casecount for) (MCASE: list Pfs. Similar to IX for CASE: ì N? IF (Valid n) 2 + SWAP (...⊭fa+2 n) ELSE DUP CO >R 6 + SWAP DROP R> (... Pfs+6 c) THEN 1 - DUP + + (... pfa+k+2En-11) @ EXECUTE ; : PSUEF (Pfa adrdef...) (Counts # cases entered and stores) (in casecount at Pfa+1 . The address of the default of a is) (at adridef = pfa+6+2[n-1]) OVER 6 + - 2 / (... pfa n-1) 1+ SWAP 1+ C! # 1 INSTALL PSFIND ATKIN DDESPART 1 INSTALL PEDER ATMIN [DEF] (POSITIONAL TYPE WITH HIGH LEVEL BEF IN LIST, TYPE 2 : 2FIND (n Pfa...adrn) (Finds address, adrn , of nth high) (level code sequence. Start at Pfa of list. Return default) (code address if n<1 or n>casecount) N? 0= IF (def) >R BROP R 1+ C@ 1+ R> THEN (...n+f pfs) SWAP >R 4 + 0 BEGIN 1+ (... Ffat4 1, Save ntf) R OVER = 0= (... pfat4 1 f) WHILE (count not=n+f) (... Ffaid count) >R 2 - @ 2+ R (... adraxt count) REPEAT DROP RO DROP 1 --> : PHFIND (n pfs...) (Find and execute hl code sea n in type) (2 caselist, Pfs . Execute default if out of ranse.) 2FIND EXECUTE ; : H: (Ffa...Ffa adrl) (Begins headerless definition in an) (MCASE: word, Pfa , Compiles dummy link address, compiles) (colon and leaves address of link to be used by (H) DUP 1+ CE 1+ OVER 1+ C! (Updates casecount) BUP CE 2 HOD IF (odd) DUP CO 14 OVER C! THEN (Updates type) HERE EOL . (Temporary link) COLON , ; IMMEDIATE : iH (pfa adrl...pfa) (Terminates headerless definition) (besun by H: . Adjusts link, compiles (S) HERE 2+ SWAP ! EDL ; ; IMMEDIATE = -i: PHDEF (pfa adridef...) (Besins headerless defin of) (default. Compiles COLON) DROP DROP COLON , ; 2 INSTALL PHEIND ATKIN DOESPART 2 INSTALL PHDEF ATKIN [DEF]

(3)	Range type	
	MCASE: xxxx RANGE [Ll , Hi ,] H:seq1 ; [L2 , H2 ,] H:seq1 ;	H ;H
	 [Ln , Hn ,] H:seqn : DEFAULT:seqd ;	;H

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.

175

175

175

175

175

175

175

175

176

176

178

176

176

176

176

177

177

177

177

177

177

177

177

177

177

177

178

178

178

178

178

178

178

Examples

of all 6 possible Examples combinations of case structures are given on Screens 180 and 181. If the screen is loaded and examples tested, typical execution results should be:

EXECUTE	RESULT TYPED
1 BARN	MOO
2 BARN	OINK
18 BARN	PEEP (Default)
1 200	PEEP PEEP PEEP
3 200	PEEP PEEP MOO
-6 200	OINK OINK OINK
(Default	:)
1 FARM	OINK (Default)
77 FARM	MOO
-10 CASE (Default	MOOOINK PEEP
77 CASE	[~] MO0000000
-10 CORRAL	PEEP PEEP
-1 CORRAL	OINK OINK
309 CORRAL	PEEP OINK MOO
310 CORRAL	MOO (Default)

COMMENTS

- 1. Kim Harris' case compiler, CASE: avoids the use of OVER = IF DROP ELSE ... THEN for every case as used in many of the other CASE constructs. The result is shorter compiled code in The compiler, the application. 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

: XEGARRAY CASE: ; : XEQVAR (LIST DOES) @ EXECUTE ; 174 6 174 7 174 8 : KSDEF (pfs adrdef...) (Counts # cases entered and stores) 174 9 174 10 (in casecount at pfa+1. Address of default ofa is) (adr def=#fa+6+41n-13) 174 11 OVER 6 + - 4 / 1+ SWAP 1+ C! # 174 12 174 13 : KSFIND (K pfa...) (Searches type 3 list for match of Key) 174 14 (to K . Starts at pfa+2 . Executes of a after matched) ---> 174 15 (Key of default if no match found.) 175 0 24 BEGIN 1 >R DUP 8 EOL -1 (... k adr1 f) IF (not ECL) OVER OVER 2 = (...K adm1 k=27) 2 3 IF (matched) 2+ (...K ad:1+2) ELSE RD 1 - DR 4 + THEN (... k adenxt) 4 5 ELSE (EOL) 2+ (...K adridef) THEN R> 6 (... K adrnew f) UNTIL (Matched of EOL) SWAP DROP @ EXECUTE ; 7 8 175 9 3 INSTALL ASFIND ATKIN DOESPART 3 INSTALL KSDEF ATKIN EDEFD 175 10 175 11 175 12 (KEYED OPTION WITH HIGH LEVEL DEF IN LIST, TYPE 4 ۱ 175 13 175-14 : KHFIND (K pfa...) (Sparches type 4 list for match of kes , 175 15 (to K . Starts at Pfa+2 . Executes high level sequence) ---(following match or default sequence if no match found,) 175 0 176 1 2+ BEGIN 1 >R BUP @ EOL -(...K adrif) IF (not EOL) OVER OVER @ = (...K ed:1 k=27) 2 IF (matched) 4 + 3 (+++k adr1±4) ELSE RD 1 - DR 2+ @ THEN (...k sornxt) 4 ELSE (EOL) 2+ -5 (...K adrdef) THEN R> 6 (... K admosu f) 7 UNTIL (Matched of EOL) SWAP DROP EXECUTE ; 8 176 9 4 INSTALL KHEIND ATKIN DOESPART 4 INSTALL PHOSE ATKIN [DEF] (Same as type 2) 176 10 176 11 176 12 (RANGE TYPE WITH SINCLE WORD EXECUTION OPTION, TYPE 5 ...) 176 13 176 14 : RSDEF (pfa adridef...) (adridef= pfa+6+6En+13 Compute n and) 176 15 (store at Ffat1) OVER 6 + - 6 / 1+ SWAT 1+ CE # G 1 : RANGE? (K adr...f) (True if K>= value at adr AND KK= value) 177 2 177 **3** 177 **4** (at ad: +2) 4 >R BUP R @ < SWAP R> 2+ @ > DR 0= ; 177 5 177 6 : RSFIND (K pfa...) (Searches type 5 list for first occurrent 7 I be of K within pair of ranse values. Executes of a follow- / 8 (ins pair. Executes default of a if not found) 1 >R DUP @ EOL - (...K adr1 f) IF (not EOL) OVER OVER RANGE? (...K adr1 g) 9 24 BEGIN 1 DR DUP @ EOL -177 10 IF (in ranse) 4 + - 11 (...K adr1+4) 12 ELSE R> 1 - >R 6 + THEN (...k adinxt) ELSE (EOL) 2+ (...k adridef) :3 14 THEN R> (...K adrnew f) UNTIL (In ranse or EOL) SWAP DROP @ EXECUTE ; 15 0 1 5 INSTALL RSFIND ATKIN DOESPART 2 5 INSTALL RSDEF ATKIN CDEF1 - 3 4 (RANGE OPTION WITH HIGH LEVEL DEF IN LIST, TYPE 6 - 5 : RHFIND (K pfa...) (Searches type 6 list for first occurr-) 178 6 178 7 (ence of K within pair of ranse values. If found, executes) 8 (following high level sequence: else executes def sequence) 178 9 2+ BEGIN 1 >R BUP @ EOL -1 >R DUP @ EOL - (...K adr1 f) IF (not EOL) DVER OVER RANGE? (...K adr1 s) 178 10 178 11 IF (in ranse) 6 + (...k adr1+6) ELSE R> 1 - >R 4 + @ THEN (...k admnxt) 178 12 178 13 ELSE (EOL) 2+ (...k adrdef)

are fig-FORTH functional equivalents of McCourt's definitions. Hence CASE: can be used as an Execution Array as suggested by McCourt. The definitions of AT, ATKIN and INSTALL on screens 167 and 168 can be used ala McCourt to change the elements in CASE: list words. They are used in the form

k INSTAL yyyy ATKIN XXXX

to change the k'th element in a case list, xxxx defined by CASE: to the code field address of yyyy. Now whenever k xxxx is encountered, the word yyyy will be executed rather than the original word in the k'th position of the case list.

Using the previous CASE: example of BARN, if we execute

2 INSTALL COW ATKIN BARN

the second case in BARN will be changed from PIG to COW. Later execution of 2 BARN anywhere in the program will then type MooOOoo instead of Oink.

Although this is non-structured programming, it is still a valuable programming tool when used properly. The present definitions of INSTALL and ATKIN can be used within a color definition.

Please note that the use of the Execution Array in the development of MCASE: on screen 169 is purely stylistic. It is not a necessary feature of the development.

3. The essentially unique feature of FORTH is that it is extendable by the user. With an expanding FORTH literature, it is clear to this author that FORTH will improve with time faster than all other languages and that there is no upper limit to its improvement. It has been less than 18 months since I first got FORTH up and running. In that short period of time, thanks to the fig literature, the FORTH system I have running now is, in my opinion, vastly superior to any other language I have ever seen. And it will get better!

178 14 THEN R> Clinik admines f) UNTIL (In range of EDL) SWAP DROP EXECUTE # 178 15 179 0 179 1 179 2 179 3 6 INSTALL RHFIND ATKIN DOESPART & INSTALL PHDEF ATKIN [DEF] (Same as tures 2 and 4) 31 179 45 179 179 6 179 7 179 8 179 9 179 10 179 11 179 12 179 13 177 14 179 15 100 (MOADE: EXAMPLES) Ĝ 189 1 : PIC ." OINK " # : COW ." MOD" # 125 2 100 3 : CHICK .* PEEP * 4 180 4 180 MCASE: BARN COW PIG CHICK DEFAULT: CHICK # 185 6 180 7 HEASE: 200 H: CHICK CHICK CHICK IH 180 8 IK 9 H: COW ." 035933" 183 H: CHICK CHICK COW 185 10 i H 189 11 DEFAULT: PIG PIG PIG # 189 12 MCASE: FARM KEYED [8) + 3 PIG 182 13 C 77 + 3 COW 163 14 189 15 [67 +] CHICK --'> BEFAULT: FIG # 181 0 181 1 MCASE: CAGE NEYED [77 + 3 H: COW +" 000000" ÷Η 181 C 8) , J H: PIG FIG 3 iΗ 181 E 67 .] H: CHICK CHICK 181 4 ŧΗ DEFAULT: COU FIG CHICK # 5 181 181 ó HCASE: PEN RANGE [-32768 + -1 + 3 CHICK 181 7 8 ο, 1 + 3 FIG 181 Ľ 1 + 32767 + 3 COW 161 9 ſ DEFAULT: # 181 10 181 11 MCASE: CORRAL RANGE [-10 + -5 +] H: CHICK CHICK iΗ 101 17 C -1 + -1 + 3 H: PIG SPACE FIG #H C 0 + 0 + 3 H: COW SPACE COW #H 181 13 181 14 [1 + 307 +] H: CHICK PIG COW #H DEFAULT: COW # 181 15

FORTH STANDARDS TEAM MEETING

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 4H Center, a selfcontained educational facility, just outside Washington, DC. The campus-like Center has meeting rooms, dining facilities and dormitory 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 (other than team members) must submit a biography by April 15 for consideration by the credentials committee. You should include:

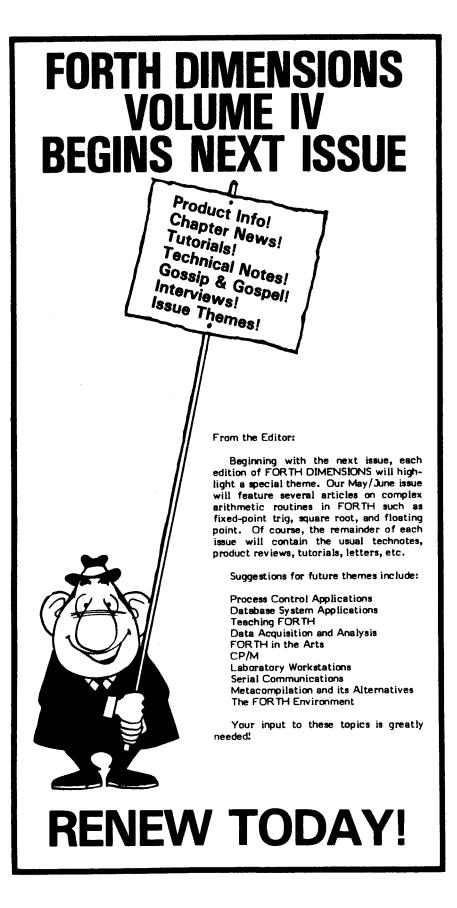
- Your skills and comprehension of multiple FORTH dialects and their application.
- 2. Why your views are representative of a significant portion of the FORTH community.

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



LECTURES ON APPLIED FORTH

a two day seminar 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 Bases and Process Control there will be a two day seminar on Applied FORTH. Managers and programmers will find these lectures very useful for exploring FORTH applications and programming concepts. Each lecturer will also lead a Working Group at the subsequent Conference. Participants should have a copy of Leo Brodie's book. Starting FORTH, 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".

Kim Harris, of Laxen & Harris, Inc., on "FORTH Programming Style".

Hans Nieuwenhuijzen, of the University of Utrecht, on "FORTH Programming Environment"

Larry Forsley, of the Laboratory for Laser Energetics, on "Extensible Control and Data Structure".

David Beers of Aregon Systems, Inc., on "A Large Programming Project Case Study: Building a Relational Database in FORTH".

Steven Marcus of Kitt Peak National Observatory, on "Assemblers & Cross Assemblers".

James Harwood of the Institute for Astronomy at the University of Hawaii, on "Computation Tradeoffs".

Roger Stapleton of St. Andrews Observatory, Scotland, on "Hardware Control with FORTH"

Raymond Dessey of Virginia Polytechnic Institute, on "Concurrency, Networking and Instrument Control".

> REGISTRATION FORM (must be received by April 23, 1982)

Name		
Address		
City	StateZIP	
Phone (Days) ()		
	CHOICES TO BE MADE	
Applied FORTH Semi	inar, May 17 & 18	\$200.00
1982 Rochester FOI	RTH Conference, May 19-21	100.00
Housing for: (circ	cle dates) May 16 17 18 19 20 21	<pre>\$ 13.00/person dbl 16.50/person sgl</pre>
	TOTAL AMOUNT ENCLOSED \$	-
Make checks payable to	: "University of Rochester/FORTH Con:	ference"

Send check and Registration to:

Mrs. B. Rueckert, Lab for Laser Energetics, 250 E River Rd, Rochester, NY 14623 For information call: Barbara Rueckert (716) 275-2357

NEW PRODUCTS

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 by 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

FORTH Programming Aide

FORTH Programming Aids are high level FORTH 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 obtAin 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 3K bytes and a recommended 13K bytes of free dictionary space. \$150 single CPU license; \$25 for manual alone (credit applied toward program purchase). California 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.

Ben Curry Curry Associates PO Box 11324 Palo Alto, CA 94306

New Book: Introduction to FORTH

Introduction to FORTH, a 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 learn how to write computer software using FORTH.

No previous knowledge of FORTH is required, but some exposure to Microsoft Level II BASIC will be heipful. 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.

RENEW NOW!

FORTH for Ohio Scientific

We've received from Technical Products Co. a copy of their newsletter. This issue contains product news and update acreens for FORTH-79. We applaud their intent of good customer support, but note technical errors in definition of several standard words (WORD, R@, END-CODE, 2CONSTANT, DK). This OSI-FORTH operates with Ohio Scientific OS_65D 3.3 operating system release.

Their new address is Technical Products Co., Box 2358, Boone, NC 28607--ed.

RENEW TODAY!

MCZ, Z05, UDS FORTH

FORTH is now running on Zilog MCZ, ZDS, and Multitech UDS microcomputer systems. It has compiler, editor, assembler, 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.

Software for OSI C1P

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. (Drills basic 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. 1 \$11.95 (specify grade level)

Want Ads Life Skills - A program that helps slow readers understand the Want Ads. Five levels of difficulty. \$7.95

Rescue Ship - Transport injured soldiers to the hospital. But the enemy has covered the ocean with mines. One of them could destroy you.

Addition - \$11.95 Subtraction - \$11.95 Multiplication - \$11.95 (all three on tape - \$28.00)

Please include \$1.00 to cover postage and handling and send to:

Henry Svec 668 Sherene Terrace London Ontario Canada N6H 3K1 The following vendors have versions of FORTH available or are consultants. (FIG makes no judgment on any products.)

ALPHA MICRO Professional Management Services 724 Arastradero Rd. #109 Palo Alto, CA 94306 (408) 252-2218

Sierra Computer Co. 617 Mark NE Albuquerque, NM 87123

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

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

George Lyons 280 Henderson St. Jersey City, NJ 07302 (201) 451-2905

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

CROSS COMPLERS Nautilus Systems P.O. Box 1098 Sente Cruz, CA 95061 (408) 475-7461

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

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

M & B Design 820 Sweetbay Drive Sunnyvale, CA 94086

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

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

PDP-11 Laboratory Software Systems, Inc. 3634 Mandeville Canyon Rd. Los Angeles, CA 90049 (213) 472-6995

OSI Consumer Computers 8907 LaMesa Bivd. LaMesa, CA 92041

(714) 698-8088

FORTH VENDORS

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

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

Tom Zimmer 292 Falceto Dr. Milpites, CA 95035

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

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

TRS-80 The Micro Works (Color Computer) P. O. Box 1110 Dei Mar, CA 92014 (714) 942-2400

Miller Microcomputer Services 61 Lake Shore Rd. Natick, MA 01760 (617) 653-6136

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

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

6502 Eric C. Rehnke 540 S. Ranch View Circle #61 Anaheim Hills, CA 92087

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

8080/Z80/CP/M Leboratory Microsystems 4147 Beethoven St. Los Angeles, CA 90066 (213) 390-9292

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

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

68000

Emperical Res. Grp. P. O. Box 1176 Milton, WA 98354 (206) 631-4855

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

Forward Technology 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 Products Interactive Computer Systems, Inc. 6403 Di Marco Rd. Tampa, FL 33614

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

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

Consultants Creative Solutions, Inc. 4801 Randolph Rd. Rockville, MD 20852

Dave Boulton 581 Oakridge Dr. Redwood City, CA 94062 (415) 368-3257

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

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

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

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

Microsystems, Inc. 2500 E. Foothill Blvd., #102 Pasadena, CA 91107 (213) 577-1471

VENDORS: FORTH DIMENSIONS will go to a product matrix in Volume IV. Send in a list of your products and services by April 18

FIG CHAPTERS

How to form a FIG Chapter:

- You decide on a time and place for the first meeting in your area. (Allow at least 8 weeks for steps 2 and 3.)
- Send FIG a meeting announcement on one side of 8-1/2 x 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 am.

Southern California

Los Angeles

4th Sat FIG Meeting, 11:00 a.m., Allstate Savings, 8800 So. Sepulveda, L.A. Philip 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. Guy Kelly, (714) 268-3100, x 4784 for site.

Northwest

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

New England Boston

Ist 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.m., Cochituate, MA. Dick Miller, (617) 653-6136 for site.

Southwest

Tulsa

3rd Tues FIG Meeting, 7:30 p.m., The Computer Store, 4343 So. Peoria, Tulsa, OK. Bob Giles, (918) 599-9304 or Art Gorski, (918) 743-0113.

Austin John Hastings, (512) 327-5864.

Dallas Ft. Worth

4th Thur

ur FIG Meeting, 7:00 p.m., Software Automation, 1005 Business Parkway, Richardson, TX. Marvin Elder, (214) 231-9142 or Bill Drissel (214) 264-9680.

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

Midwest

Detroit Dean Vieau, (313) 493-5105.

Minneaota 1st Mon FIG Meeting. Mark Abbott (daya), (612) 854-8776 or Fred Olson, (612) 588-9532. Call for meeting place or write to: MNFIG, 1156 Lincoln Avenue, St. Paul, MN 55105.

Foreign

Australia Lance Collins (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, Minato-ku, Toyko, 105 Japan.
- Canada Guebec Gilles Paillard, (418) 871-1960 or 643-2561.
- W. Germany Wolf Gervert, Roter Hahn 29, D-2 Hamburg 72, West Germany,(040) 644-3985.

SIGN UP A FRIEND START A FIG CHAPTER PUT THE ORDER FORM ON THE BULLETIN BOARD

-

Phoenix Peter Bates at (602) 996-8398.