Ipso Facto

ISSUE 38 37 OCTOBER 1983

INDEX PAGE

A PUBLICATION OF THE ASSOCIATION OF COMPUTER-CHIP EXPERIMENTERS (ACE) 1981

Executive corner	2
Member's corner	3
Errata	4
Murphy's Laws of computing	9
The Last Memory	13
A VLIST Fix for FIG-FORTH	15
A Simple 8 bit Multiply Program	17
An 1802 Assembler - FORTH style	19
Some Hints on Program Debugging	27
FIG-FORTH Expansion	31
FORTH and the Smarterm - 80	35
1802 Computer Products Available from John Ware	39
Club Communique	41

IPSO FACTO is published by the ASSOCIATION OF COMPUTER-CHIP EXPERIMENTERS (A.C.E.), a non-profit educational organization.

1983-1984 EXECUTIVE OF THE ASSOCIATION OF COMPUTER CHIP EXPERIMENTERS

President:	John Norris 416-239-8567	**Vice-President:**	Tony Hill 416-876-4231
Treasurer:	Ken Bevis 416-277-2495	**Secretary:**	Fred Feaver 416-637-2513

Directors: Bernie Murphy - Fred Pluthero - John Norris - Mike Franklin

Newsletter:

Production Manager:	Mike Franklin 416-878-0740	**Product Mailing:**	Ed Leslie 416-528-3222 (Publication)
Editors:	Fred Feaver Tony Hill		Fred Feaver 416-637-2513 (Boards)
Publication:	Dennis Mildon John Hanson		

Club Mailing Address:

A.C.E.
c/o Mike Franklin
690 Laurier Avenue
Milton, Ontario
Canada
L9T 4R5
416-878-0740

ARTICLE SUBMISSIONS:

The content of Ipso Facto is voluntarily submitted by Club Members. While ACE assumes no responsibility for errors nor for infringement upon copyright, the Editors verify article content as much as possible. ACE can always use articles, both hardware and software, of any level or type, relating directly to the 1802 or to micro computer components, peripherals, products, etc. Please specify the equipment or support software upon which the article content applies. Articles which are typed are preferred, and are usually printed first. Please send originals, not photocopy material. We will return photocopies of original material if requested.

PUBLICATION POLICY:

The newsletter staff assume no responsibility for article errors nor for infringement upon copyright. The content of all articles will be verified, as much as possible, and limitations listed (i.e. Netronics Basic only, Quest Monitor required, require 16K at 0000-3FFF etc.). The newsletter will be published every other month, commencing in October. Delays may be incurred as a result of loss of staff, postal disruptions, lack of articles, etc. We apologize for such inconvenience - however, they are generally caused by factors beyond the control of the Club.

MEMBERSHIP POLICY:

A membership is contracted on the basis of a Club year - September through the following August. Each member is entitled to, among other privileges of Membership, all six issues of Ipso Facto published during the Club year.

The Second Annual 1802 Computer Conference

- by Fred Feaver, ACE Secretary

The second annual 1802 Computer Conference was held in the Sheridan Hall Theatre of Sheridan College in Oakville, Ontario on Saturday August 20, 1983.

The attendance was approximately the same as last year and those in attendance asked many intelligent and interesting questions of the speakers.

The welcome message to the Conference and introduction of the speakers was capably handled by Fred Pluthero, the Conference Chairman.

The first speaker was Jim Greer of Computer Systems, Sheridan College, speaking on Teledon, Teletext and Videotext.

Mr. Greer gave an outline of the history of Teledon and similar systems starting about 1970 and interlinking its history with developments in Britain, Canada and the U.S.A. He described some uses, such as ordering goods from displays on a home television screen.

Several other speakers kept up the interest of the audience.

Mr. Jack Neil from Montreal General Hospital spoke on Software Switching on Medical Data in hospitals. He spoke on the need for such a switching system and showed schematics of hardware used with the system.

Mr. Pierre Andeweg, RCA Field Applications Engineer, outlined several new products being produced by RCA, such as OMOS chips and CMOS versions of the 6502 and other popular CPU's.

Mr. John Langtrey of Bell Telephone Co. together with Dave Kerr and the assistance of other radio amateurs in the "field" gave a demonstration of the use of PACKET DATA TRANSMISSION between two Ham radio stations.

Dr. NcSolntseff of Computer Science Department of McMaster University, Southern Ontario Forth Interest Group, spoke on Programming with Forth.

Our own Club Software Co-ordinator Wayne Bowdesh talked on designing a DOS.

Mr. Dan Thomas spoke on Micro Forum Intelligent Control of Portable Traffic Lights using 1802 microprocessors to control traffic lights which in turn controlled traffic during construction of a highway.

Probably the piece de resistance of the meeting was the drawing for the two door prizes - Videotex 3801 Teletex terminals donated by RCA. The winners were: James Tote, Forklift Systems Inc., 9313 Worrel Avenue, Lanham, Mo. 20706 and Harry Lidkea c/o Optical Business Machines, 804 West NewHaven Avenue, Melbourne Fl. 32901.

Many other donations such as joysticks, headphones, a 6800 text and other handbooks were raffled off.

These donations were contributed by: Active Surplus, 347 Queen Street West, Toronto; Arkon, 407 Queen Street West, Toronto; Ace Computer Supply Inc., 329 Queen Street West, Toronto; Surplustronics, 310 College Street, Toronto; Western Radio Supply Co. Ltd., 182 Rebecca Street, Hamilton; and L.A. Varah, 505 Kenora, Hamilton.

We thank all these companies for their support.

We hope that the next conference will be just as interesting as this one and that you will attend.

Members Corner

For Sale: T. Piper, 2555 Cote Vertu #100, St. Laurent, P.Q., Canada, H4R 1Z6
phone: 514 - 341 - 6780 ext. 567 (off) or 514 - 336 - 0880

Quest Super ELF, with ACE Quest expantion board, 32K DRAM board, 16K Eprom board. ACE backplane II, Ipso Facto # 19 - 36, Defacto, Quest Data # 1-24. RCA manuals price negotiable.

For Sale: D. Moyer, 1856 Pacific Ave, Winnipeg, Man, Canda, R2R 0G1

Netronics Elf II, 1 - 4k Ram board, 1- giant board, 1 - light pen
Assembly manuals and "A Short Course in Programming"
$200.00 CDN or US, includes shipping.
ACE VDU board - unused $37.00

Information: H. Shanko, 15025 Vanowen St. #209. Van Nuys, Ca. USA, 91405

1802 cross-software notice

Pass the word - to anyone with access to an Atari 400 or 800 - there is software, via Atari's APX (Atari Program Exchange, 3rd party software), 'COSMATIC ATARI DEVELOPMENT PACKAGE' (APX-20051) for the 1802. It provides an 1802 cross-assembler and a Development System set of software, occupying about 600 of 720 sectors.

I will shortly be using it and will report on its usefulness if there is interest in this product.

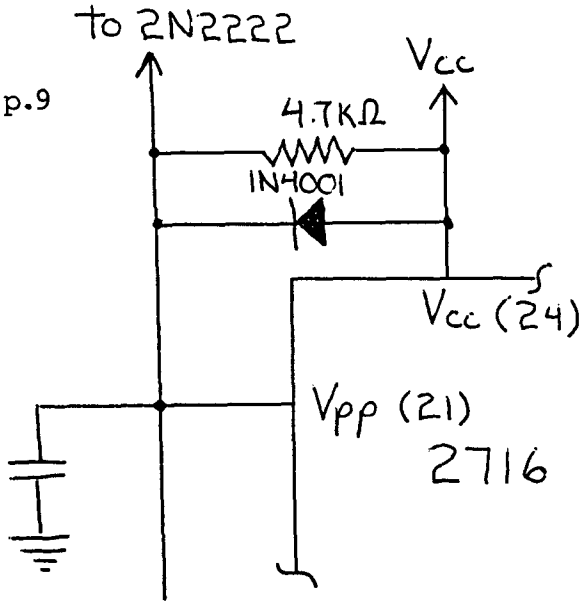
ERRATA

Alien - L.Owen, Ipso Facto #35, p.14

corrections: M - 0124 - 29
0202 - 96
040F - D2
04F0 - 35

A Simple Eprom Programmer - D. Schuler, Ipso Facto #36, p.9

schmatic error, revision below:



ACE VDU Board

add - 22k ¼ watt resistor on back of board between pins 13 and 16 of 4025.
add - jumper betweem S1 d6 plate through hole and plate through hole 1 inch above connected to SIP pin 4 (74C244 pin 4)
cut - trace between S1 d6 plate through hole and plate through hole ½ inch above connected to Sip pin 7 (74C244 pin 7)

"WANTED - An A.C.E. 2708 EPROM Board. Cash or will trade a Technitrol UV EPROM Eraser"

W.A.Erskine
131 Ave. Adobe
San Clemente, CA 92672
USA

(714) 492-9798

8/14/83

Association of Computer Experimenters
c/o M.E. Franklin
690 Laurier Ave.
Milton, Ontario
Canada L9T 4R5

After 2 years in seclusion (ask my wife about that), I have finished development of a Disk Operating System for the 1802. It will support either 1 or 2 5-1/4 inch drives. It is sort of a CP/M look-alike in the commands, although not quite as sophisticated. The system is comprised of a disk controller card, which uses the DMA function of the 1802 to transfer data from a 1771 type controller chip. It is physically compatible with the RCA 44 pin bus, but could be adapted to any 1802 system, including the ACE. John Deering of the Elf of the Valley club is doing this now. The software resides in two 2716 EPROMS on the board, and an additional socket is provided for a system monitor. The DOS software is located at addresses E000-EFFF, and the monitor socket at F000. It is completely I/O independent, as all I/O is done through vectors located in RAM, which is loaded from the disk when booted. It uses named files, with dynamic disk space allocation, up to 75K bytes on each diskette. Built-in commands are SAVE a file, ERAse a file, RENname a file, COPY a file, change drive, DIRectory display, SYS a new diskette, and NEW (to initialize a directory). ".COM" type files are run automatically by typing their name.

The minimum hardware requirements are 8K of RAM, an ASCII keyboard and video display (preferably 64-80 characters), and no other DMA functions used at the same time. No I/O ports are used, and only 1 flag (EF1) is required by the system. 256 bytes of memory are needed at FF00-FFFF, and is on the controller board. The controller is memory-mapped at addresses FEF8-FEFB. The system requires +5V, +12V, and either -5V or -8 to -20V. A -5 regulator can be installed on the board, if required. The disk drives will require their own separate power supply of +5 and +12. It will not support 8 inch drives without extensive software changes, which I am not going to do. All internal subroutines are described in the documentation.

I am prepared to offer the system as a blank PC board with the two 2716's, system diskette with Format, Disk Dump and Length utility programs. Full documentation is included on bringing the system up and operating it. A text editor and assembler are almost completed, and will be available soon. The first batch of PC boards have already been sold to our club members, and more will be made when enough people have ordered them to make it feasible.

The price for the system is $75.00 US. Retail prices for the required components to complete the board is about $85.00, not including disk drives. A copy of the Instrucion Manual is available for $5.00, credited toward purchase of the system.

Richard M. Cox
2670 Calle Abedul
Thousand Oaks, Calif. 91360

CORRECTIONS FOR HIGH RESOLUTION LINE GEN PROGRAM

I INADVERTANTLY SENT THE VERSION WITH MACRO ROUTINES FOR MY SYSTEM. IT WAS ASSEMBLED AT E100 INSTEAD OF 0C00 AS IT SHOULD BE. ALSO THERE ARE ROUTINES IN IT NOT NEEDED BY LINEGEN.

REQUIRED ROUTINES ARE - MACRO, SAVAE, PNTAD, NEG2 & RSTCA
BE SURE TO KEEP CODE WITHIN THESE SUCH AS MUL & NEG1.

I WOULD LIKE TO HERE FROM ANYONE WHO TRIED IT.

LYNN KEENLISIDE
519-471-8688

```
1                                       ;MACRO SUBROUTINES
2                                       ;FOR LINEGEN PGM.
3                                       ;
4                                       ;CALLING SEQUENCE IS-
5                                       ;SEP R7   .BYTE
6                                       ;WHERE .BYTE IS THE SUB #
7                                       ;THIS MAY BE FOLLOWED BY
8                                       ;ANY INLINE BYTES REQUIRED
9                                       ;
10                                      ;
11      0C00                            .= $0C00        ;START
12                                      ;
13  0C00  9F            FINI:   GHI R15         ;RETURN WITH D AS BEFORE
14  0C01  D3                    SEP R3          ;RETURN TO CALLER
15  0C02  BF            MACRO:: PHI R15         ;KEEP D FOR RETURN
16  0C03  43                    LDA R3          ;GET SUB #
17  0C04  A7                    PLO R7          ;BR TO VECTOR
18  0C05  C0  0C  00    NOP:    LBR FINI        ;DUMMY
19  0C08  30  00        NOP1:   BR FINI         ;DUMMY
20  0C0A  30  20        SAVAE:: BR VSAVAE       ;SAVE REG A TO REG E VIA X
21  0C0C  30  00                BR FINI         ;DUMMY
22  0C0E  30  00                BR FINI         ;DUMMY
23  0C10  30  00                BR FINI         ;DUMMY
24  0C12  30  00                BR FINI         ;DUMMY
25  0C14  30  45        PNTAD:  BR VPNTAD       ;POINT ADDRES OF X,Y
26  0C16  30  7A        NEG2:   BR VNEG2        ;NEGATE 2 BYTES VIA RC
27  0C18  30  00                BR FINI         ;DUMMY
28  0C1A  30  00                BR FINI         ;DUMMY
29  0C1C  30  36        RSTCA:  BR VRSTCA       ;RESTORE REG'S C TO A
30      0C20                    .= $0C20        ;BEGIN MACROS
31  0C20  9A            VSAVAE: GHI R10
32  0C21  73                    STXD
33  0C22  8A                    GLO R10
34  0C23  73                    STXD
35  0C24  9B                    GHI R11
36  0C25  73                    STXD
37  0C26  8B                    GLO R11
38  0C27  73                    STXD
39  0C28  9C                    GHI R12
40  0C29  73                    STXD
41  0C2A  8C                    GLO R12
42  0C2B  73                    STXD
43  0C2C  9D                    GHI R13
44  0C2D  73                    STXD
45  0C2E  8D                    GLO R13
46  0C2F  73                    STXD
47  0C30  9E                    GHI R14
48  0C31  73                    STXD
49  0C32  8E                    GLO R14
50  0C33  73                    STXD
51  0C34  30  00                BR FINI
52  0C36  60            VRSTCA: IRX             ;MOVE UP TO DATA
53  0C37  72                    LDXA
54  0C38  AC                    PLO R12
55  0C39  72                    LDXA
56  0C3A  BC                    PHI R12
57  0C3B  72                    LDXA
58  0C3C  AB                    PLO R11
59  0C3D  72                    LDXA
60  0C3E  BB                    PHI R11
61  0C3F  72                    LDXA
62  0C40  AA                    PLO R10
63  0C41  F0                    LDX
64  0C42  BA                    PHI R10
65  0C43  30  00                BR FINI
```

```
 66   0C45   F8   13        VPNTAD: LDI YOFF        ;POINT TO YOFFSET
 67   0C47   AC                    PLO R12
 68   0C48   EC                    SEX R12
 69   0C49   12                    INC R2           ;POINT TO Y
 70   0C4A   42                    LDA R2           ;GET IT FROM STACK
 71   0C4B   F4                    ADD              ;ADD Y OFFSET
 72   0C4C   AA                    PLO R10          ;MOVE IT TO R.A
 73   0C4D   2C                    DEC R12          ;POINT TO XOFFSET
 74   0C4E   02                    LDN R2
 75   0C4F   F4                    ADD
 76   0C50   52                    STR R2
 77   0C51   FA   07               ANI $07
 78   0C53   AC                    PLO R12
 79   0C54   0C                    LDN R12          ;GET BIT MASK
 80   0C55   AB                    PLO R11
 81   0C56   02                    LDN R2           ;GET X BACK
 82   0C57   FB   FF               XRI $FF          ;COMPLIMENT
 83   0C59   F6                    SHR              ;& DIVIDE
 84   0C5A   F6                    SHR              ;BY 8
 85   0C5B   F6                    SHR
 86   0C5C   52                    STR R2
 87   0C5D   F8   05               LDI $05          ;MULTIPLY BY 32
 88   0C5F   A8                    PLO R8           ;THAT'S 1 LINE UP
 89   0C60   F8   00               LDI $00          ;FOR EVERY Y POINT
 90   0C62   BA                    PHI R10
 91   0C63   8A             MUL:   GLO R10          ;BEGIN MULTIPY
 92   0C64   FE                    SHL
 93   0C65   AA                    PLO R10
 94   0C66   9A                    GHI R10
 95   0C67   7E                    SHLC             ;THIS GET'S ADDRESS OF Y
 96   0C68   BA                    PHI R10
 97   0C69   28                    DEC R8
 98   0C6A   88                    GLO R8
 99   0C6B   3A   63               BNZ MUL          ;LOOP TILL DONE
100   0C6D   F8   14               LDI VSTART       ;POINT TO VSTART
101   0C6F   AC                    PLO R12
102   0C70   E2                    SEX R2
103   0C71   8A                    GLO R10
104   0C72   F4                    ADD              ;ADD X TO ADDRESS
105   0C73   AA                    PLO R10
106   0C74   EC                    SEX R12
107   0C75   9A                    GHI R10          ;ADD CARRY & VSTART
108   0C76   74                    ADC

109   0C77   BA                    PHI R10
110   0C78   30   00               BR FINI
111   0C7A   F8   01         VNEG2: LDI $01
112   0C7C   73                    STXD
113   0C7D   C8                    LSKP
114   0C7E   43              VNEG1: LDA R3
115   0C7F   AC                    PLO R12
116   0C80   0C                    LDN R12
117   0C81   FD   00               SDI $00
118   0C83   5C                    STR R12
119   0C84   30   01               BR FINI +1
120          0D00                  .= $0D00         ;BIT MAP
121   0D00   80   40         BITMAP: .WORD $8040
122   0D02   20   10                 .WORD $2010
123   0D04   08   04                 .WORD $0804
124   0D06   02   01                 .WORD $0201
125          0D10                  .=.+8            ;DATA TABLE
126   0D10   00   00         DTAB:   .WORD $0000
127   0D12   00              XOFF:   .BYTE $00       ;XOFFSET
128   0D13   00              YOFF:   .BYTE $00       ;YOFFSET
129   0D14   E8              VSTART: .BYTE $E8       ;START ADDRESS OF VIDRAM
130   0D15   00              WRTFLG: .BYTE $00       ;WRITE FLAG
131   0D16   00                      .BYTE $00
132   0D17   00   00         DELTX:  .WORD $0000     ;DELTA X & SIGN
133   0D19   00   00         DELTY:  .WORD $0000     ;DELTA Y & SIGN
134   0D1B   00   00         ERTRM:  .WORD $0000     ;ERROR TERM
135   0D1D   00   00         LAST:   .WORD $0000     ;LAST ADDRESS TO DO
136   0D1F   00              MASK:   .BYTE $00       ;LAST BIT TO DO
137          0000                  .END
```

MURPHY'S LAWS OF COMPUTING

Compiled by:
Gary Jones
7717 N. 46th Drive
Glendale, Arizona 85301

Computer hobbyists are indebted to the famous "Murphy" who first formulated the well known law: "If Anything Can Go Wrong, It Will!" and the first corollary; "At the Worst Possible Time!" Years of experience have inspired some other truisms related to the computer hobby:

*** SHAW'S PRINCIPLE ***
Build a system that even a fool can use, and only a fool will want to use it.

*** JOHNSON'S FIRST LAW ***
When any mechanical contrivance fails, it will do so at the most inconvenient time.

*** JOHNSON'S FIRST LAW AS APPLIED TO COMPUTERS ***
The computer will always go down at the most inconvenient time, especially when you have not recently backed up your data.

*** FERNI'S LAW OF GRAVITY ***
Any small part dropped while working on a power supply will never hit the ground.

*** SATTINGER'S FIRST LAW ***
Anything can be made to work, if you fiddle with it long enough.

*** SATTINGER'S SECOND LAW ***
It works better if you plug it in.

*** STURGEON'S LAW ***
95% of everything is crud.

*** HOWE'S LAW ***
Everyone has a scheme that will not work.

*** GINSBERG'S THEORUM ***
1. You can't win.
2. You can't break even.
3. You can't even quit the game.

*** GREY'S LAWS OF COMPUTER PROGRAMMING ***
1. Any given program, when running, is obsolete.
2. Any given program costs more and takes longer.
3. If a program is useful, it will have to be changed.
4. If a program is useless, it will have to be documented.
5. Any given program will expand to fill all available memory.
6. The value of a program is inversely proportional to the weight of its output.
7. Program complexity increases until it exceeds the capability of the programmer who must maintain it.

*** TROUTMAN'S POSTULATES ***

1. If a test installation functions perfectly, all subsequent systems will malfunction.
2. Not until a program has been in production for at least six months will the most harmful error be discovered
3. Interchangeable tapes won't.
4. If the input routine has been designed to reject all bad input, an ingenious idiot will discover a method to get bad data past it.
5. Profanity is the one language all programmers know best.

*** GILBEY'S LAWS OF UNRELIABILITY ***

1. Computers are unreliable, but humans are even more unreliable.
2. Any system which depends upon human reliability, is unreliable.
3. Undetectable errors are infinite in variety, in contrast to detectable errors, which by definition are limited.
4. Investment in reliability will increase until it exceeds the probable cost of errors, or until someone insists on getting some work done.

*** LUBARSKY'S LAW OF CYBERNETIC ENTOMOLOGY ***

There's always one more bug.

*** THE SNAFU EQUATIONS ***

1. Given any problem containing "N" equations, there will always be "N+1" unknowns.
2. An object or bit of information most needed will be the least available.
3. Once you have exhausted all possibilities and fail, there will be one solution, simple and obvious, highly visible to everyone else.
4. Variables won't, constants aren't.

*** IBM'S POLLYANNA PRINCIPLE ***

Machines should work, people should think.

*** THE FIRST LAW OF REVISION ***

(The "Now They Tell Us!" Law)
Information necessitating a change of design will be conveyed to the designer (or programmer) after - and only after - the plans or the program are complete.

*** THE SECOND LAW OF REVISION ***

The more innocuous the modification appears to be, the further its influence will extend, and the more plans will have to be redrawn.

*** THE THIRD LAW OF REVISION ***

If, when completion of a design is imminent, requirements are supplied as they actually are, instead of as they were meant to be, it is always simpler to start all over.

*** GRAY'S LAW OF PROGRAMMING ***

"N+1" trivial tasks are expected to be accomplished in the same time as "N" tasks.

*** LOGG'S REBUTTAL TO GRAY'S LAW ***

"N+1" trivial tasks take twice as long as "N" trivial tasks.

*** NEWTON'S APHORISM ***

A bird in the hand is safer than one overhead.

*** WEILER'S LAW OF ORGANIZATIONAL DELEGATION ***
Nothing is impossible for the man who doesn't have to do it himself.

*** WEINBERG'S MAXIM ***
If builders built the way that programmers program, the first woodpecker that came along would destroy civilization.

*** TRUMAN'S LAW ***
If you cannot convince them, confuse them.

*** JONES'S LAW ***
The man who can smile when things go wrong has thought of someone he can blame it on.

*** MURPHY'S LAW OF THERMODYNAMICS ***
Things get worse under pressure.

*** THE NON-RECIPROCAL LAWS OF EXPECTATIONS ***
Negative expectations yield negative results.
Positive expectations yield negative results.

*** THE UNSPEAKABLE LAW ***
As soon as you mention something.....
.....If it's good, it goes away.
.....If it's bad, it happens.

*** MURPHY'S MAXIM OF SELECTIVE DISPERSAL ***
What ever strikes the fan will not be distributed evenly.

*** DONALDSON'S DICTUM ***
Functioning in an organization is like pulling a dog sled, no one but the lead dog gets a change of scenery.

*** LEMARR'S POSTULATE ***
If it works, don't fix it.

*** LEMARR'S SECOND POSTULATE ***
If it jams, force it. If it breaks, it needed replacing anyway.

*** QUIGLEY'S LAW ***
Anything that begins well ends badly.
Anything that begins badly ends worse.

*** WALGREN'S SOLUTION ***
Always think of the most common problem first, not last.

*** EHRMAN'S COMMENTARY ***
1. Things will get worse before they get better.
2. Who said things would get better?

*** THE NUMBER CRUNCHER'S LEXICON ***

ASSEMBLY LANGUAGE - Profanities used by people who build computers from kits. (Such people are termed "cursors".)

BUG - A parasite which infests software. It is transmitted by illicit congress between people and programs.

CORE - The remains of an Apple when all the bytes are gone.

DATA PROCESSING - An arcane fortune telling method wherin one attempts to extract hidden meanings from numbers. Akin to numerology.

DOWN TIME - Periods when a computer is severely depressed.

ERROR - The act of buying a computer.

HEXADECIMAL - To bewitch a number.

INFINITE LOOP - See Input.

INPUT - See Throughput.

KEYPUNCH - The device that puts the little holes in keys.

LINE PRINTER - The device that puts the thin blue streaks across your notebook paper.

OUTPUT - See Input.

PERIPHERAL - Irrelevant.

REAL TIME - Whenever the computer isn't hallucinating.

REBOOT - To repeatedly kick a faulty computer until it starts working again.

TERMINAL - Said of a computer that is about to die.

THROUGHPUT - See Output.

THE LAST MEMORY

Memory chip prices have come way down since the early days of home computers, and everyone is looking for an easy way to expand their own system to its maximum memory size. Expansion memory should be easy to implement, and inexpensive, too. In addition, if one can use existing system memory, and add either RAM or ROM memory chips in accordance with one's current finances, an expansion memory board begins to look better and better.

Several companies build kit and finished 64K RAM/ROM boards for the S-100 bus, including Digital Research of Texas and WAMECO of California. However, another company, Static Memory Systems of Freeport, Illinois has a memory board which has several distinct advantages over these boards. "The Last Memory", a 64K S-100 static RAM/ROM board has the advantage of allowing the user to disable ANY 2K RAM block, to leave it empty, to use for some other memory mapped device, or to use 2Kx8 ROM chips where-ever desired.

Thus, a Super Elf owner, for example, with 4K of RAM on the Super Expansion Board, 1K of ROM at location #8000, and 1/4K of RAM in location #9800, can retain his current memory, and add RAM or ROM wherever else desired up to 64K, leaving blanks wherever his pocketbook dictates. At least this is how it appeared to me, so I sent my $95 off for the SMS Last Memory board as a RAM-less board kit. Construction was simple, the documentation easy to read, and soon I was ready to plug in a few 2Kx8 6116-3 RAMs.

Of course, it didn't work, since the Super Elf doesn't have a true IEEE-696 S-100 bus. However, with a little ingenuity, I soon was running with 32K of program RAM from #0000 to #7FFF and 8K of scratch RAM at loc #E000 to #FFFF. I can pull the memory board, and my Super Elf instantly reverts to a 4 1/4K RAM system with Super Monitor at #8000. Here's how:

1. - Jumper IC12 pin 13 to IC3 pin 5 to simulate *sW0 on S-100 pin 97.
2. - Jumper IC2 pin 10 to IC2 pin 9 to keep PDBIN at S-100 pin 78 from "floating".

3. - Jumper IC14 pin 15 to IC14 pin 14, and IC14 pin 15 to IC2 pin 13 to disable the tri-state read buffer when the Elf's 4K expansion memory is selected.

4. - Remove IC1 (74LS30) to disable the board's "FF detector". It interferes with the Super Elf data bus.

5. - Remove IC4 and IC5 (74LS138) so you don't have to worry about extended address bits 16 thru 23 and "phantom".

6. - Remove IC2K and IC4K (6116/2016 RAMs) to allow access to the Super Elf's 4K RAM.

7. - Install 2Kx8 RAM's wherever required and run a memory check test such as E.L. Smother's program (I.F. #25, Oct. '81)

I have had the Last Memory running on my system since January, adding memory chips as I could afford them. It runs great.

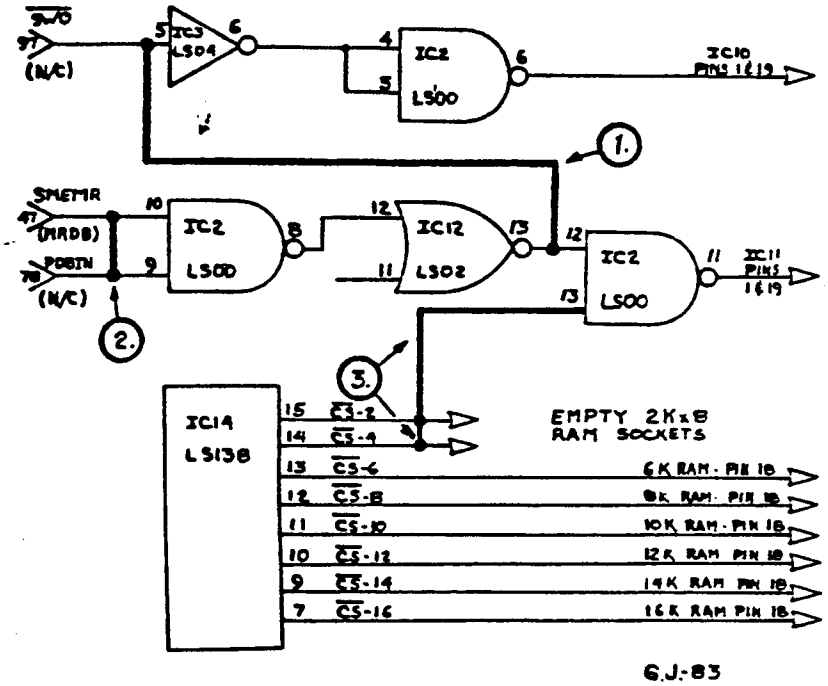
Gary Jones

Box 35172

Phoenix, Arizona 85069

THE LAST MEMORY

Changes to make it run on the Super Elf S-100 Bus.



G.J.-83

A VLIST FIX FOR FIG-FORTH

by Rick Poore, 2636 FUlbourne Dr., Cincinnati, OH 45231 USA

As pointed out by Tony HIll in IF 31, VLIST will overflow the edge of a narrow video screen. The problem seems to be with the definition of the constant C/L, which defines the screen width in characters per line. The value of this constant can only be changed indirectly, since constants in Forth are not supposed to be redefinable. The parameter field of the definition must be altered. To change this or any constant, use :

```
n ' C/L !
```

n	is the new constant value
' C/L	places the PFA of the constant on the stack
!	stores the new constant value in C/L's parameter field.

For this example, n should be the width of your output device. Be sure your number is entered in the correct base.

The output of VLIST is sloppy, since it outputs names followed only by two spaces. I wrote the definition LISTV, which clears up this and several other problems.

If your version of Forth does not support a break function, there is no way to quit a VLIST once it has started. Further, there is no way to stop the display temporarily, at least on my system. Frequently, I only need to see the first several lines of VLIST to see what words I have defined. LISTV breaks the listing in screen size pages, and waits for a key press and the end of each screen. If control-c is hit, it will quit the listing and return you to Forth.

The listing gives the original definition of VLIST, and the definition of LISTV. The definition is easily modified to fit your system. The number defined by L/S should be changed to the number of lines per screen on your terminal. The break character used to stop LISTV at the end of the page is determined in the routine PAGE. Change the value in the line KEY 3 = , where 3 is ASCII value of the key. In this case, it is control-c. The header message printed by PAGE may also be changed to suit the user.

FORTH DEFINITIONS OF VLIST AND LISTV

```
DECIMAL
: VLIST
   80 OUT !                ( force a beginning cr                          )
   CONTEXT @ @             ( get a beginning NFA for dictionary list       )
   BEGIN                   ( start a loop                                  )
     OUT @ C/L >           ( have we exceeded the limit per line?          )
       IF CR 0 OUT !       ( if so, do a cr, and reset to column 1         )
       THEN                (                                               )
     DUP ID. 2 SPACES      ( dup the NFA, and print its name field,        )
                           (   followed by two spaces                      )
     PFA LFA @ DUP 0=      ( get the next NFA and check if done            )
     ?TERMINAL OR          ( has break been hit?                           )
   UNTIL                   ( quit if either condition met                  )
   DROP                    ( drop the leftover NFA                         )
   ;

VARIABLE LINE#             ( variable to keep track of line numbers        )
24 CONSTANT L/S            ( constant for lines per screen                 )

: PAGE                     ( hold the display until a key is struck        )
  KEY 3 =                  ( wait for a key,  is it control-c ?            )
    IF ." QUIT " QUIT      ( if so, quit and print that we did so          )
    ELSE
    12 EMIT                ( do a formfeed to clear screen                 )
    ." -- FIG FORTH 1802 -- "  ( print a header across the top             )
    CR THEN                ( print a cr                                    )
    ;

: LISTV
  12 EMIT                  ( clear the screen via formfeed                 )
  80 OUT !                 ( force a cr                                    )
  CONTEXT @ @              ( get a starting NFA  for the search            )
  3 LINE# !                ( starting on line 3                            )
  BEGIN                    ( start a loop                                  )
    OUT @ C/L >            ( end of the line?                              )
      IF CR 0 OUT !        ( if so , new line                              )
      1 LINE# +!           ( increment line number                         )
      LINE# @ L/S 2 -      ( end of screen?                                )
        IF CR 60 SPACES    ( if so, print a prompt to continue             )
        ." --> " PAGE      ( prompr andwait for a key before proceeding    )
        1 LINE# !          ( reset line number                             )
        THEN               (                                               )
      THEN                 (                                               )
    DUP DUP DUP ID.        ( lots of NFA's needed, print name field        )
    PFA LFA - ABS 1+       ( determine the length of name just printed     )
    16 MOD 16 SWAP - SPACES ( print enough spaces to keep 15 wide col      )
    PFA LFA @ DUP 0=       ( end of dictionary?                            )
    ?TERMINAL OR           ( break?                                        )
  UNTIL                    ( quit if either occurred                       )
  DROP                     ( drop final NFA                                )
  ;
```

;S

SIMPLE 8 BY 8 BIT MULTIPLY PROGRAM
BY LYNN KEENLISIDE, LONDON ONTARIO

HERE'S A SHORT PROGRAM I TRANSLATED FROM AN 8080 ROUTINE AND ADDED I/O TO IT. REGISTERS ARE NOT FULLY INITIALIZED WITH HIGH BYTE AS FOR 1 MEMORY PAGE ONLY. YOU CAN ADD THIS AS DESIRED. PRODUCT IS SHOWN HIGH BYTE THEN LOW.

```
.MAIN.  CROSS - MICRO PROCESSOR ASSEMBLER 6(31)  13-AUG-83  19:06  PAGE 1
MPLY.M18

                                                ;THIS PROGRAM INPUTS 2 NUMBERS
                                                ;& OUTPUTS THEIR PRODUCT
                                                ;
                                                ;HIGH REGISTERS NOT SET
                                                ;
      0000                                      .= $0000          ;START
0000  F8  FC                                   LDI $FC           ;STACK POINTER
0002  A2                                       PLO R2
0003  F8  44                                   LDI HERE          ;DELAY SUBROUTINE
0005  A6                                       PLO R6
0006  F8  1E                                   LDI MUL           ;MULTIPLY SUBROUTINE
0008  A7                                       PLO R7
0009  E0                              START:   SEX R0
000A  64                                       OUT 4             ;DISPLAY INLINE BYTE
000B  FF                                       .BYTE $FF         ;FF READY FOR INPUT
000C  E2                                       SEX R2
000D  D6                                       SEP R6            ;CALL WAIT
000E  6C                                       INP 4             ;GET INPUT
000F  64                                       OUT 4             ;SHOW IT
0010  AA                                       PLO R10           ;HOLD IT FOR MULT
0011  D6                                       SEP R6            ;WAIT AGAIN
0012  6C                                       INP 4             ;NEXT NUMBER
0013  64                                       OUT 4             ;SHOW IT
0014  D7                                       SEP R7            ;CALL MULTIPLY
0015  D6                                       SEP R6            ;WAIT TO GIVE ANSWER
0016  64                                       OUT 4             ;SHOW HIGH BYTE
0017  D6                                       SEP R6
0018  64                                       OUT 4             ;SHOW LOW BYTE
0019  22                                       DEC R2            ;RESET STACK
001A  D6                                       SEP R6            ;WAIT BEFORE RESTART
001B  30  09                                   BR START

                                                ;
                                                ;THIS SUB ROUTINE IS AN 8 BY 8 BIT MULT
                                                ;LOW REGISTER A HAS MULTIPLIER
                                                ;D HAS MUTIPLICAND
                                                ;RESULT IN REGISTER F IS MOVED TO STACK
                                                ;THIS PART TRANSLATED FROM 8080
                                                ;
001D  D0                              END:     SEP R0            ;RETURN TO CALLER
001E  E2                              MUL:     SEX R2
001F  52                                       STR R2            ;SAVE MULTIPLICAND
0020  F8  00                                   LDI $00           ;CLEAR PARTIAL PRODUCT
0022  AF                                       PLO R15
0023  BF                                       PHI R15
0024  F8  08                                   LDI $08           ;LOOP COUNT
0026  A8                                       PLO R8
0027  8F                              LOOP:    GLO R15
0028  FE                                       SHL               ;SHIFT PARTIAL PRODUCT LEFT
0029  AF                                       PLO R15
002A  9F                                       GHI R15
002B  7E                                       SHLC              ;SHIFT CARRY ALSO
002C  BF                                       PHI R15
002D  8A                                       GLO R10
```

```
.MAIN.  CROSS - MICRO PROCESSOR ASSEMBLER 6(31)  13-AUG-83  19:06  PAGE 1-1
MPLY.M18

    55  002E  FE                       SHL             ;SHIFT MULTIPIER SAME
    56  002F  AA                       PLO R10
    57  0030  3B  39                   BNF DEC         ;DON'T ADD IF NO CARRY
    58  0032  8F                       GLO R15
    59  0033  F4                       ADD             ;ADD MULTIPLICAND TO
    60  0034  AF                       PLO R15         ;PARTIAL PRODUCT
    61  0035  9F                       GHI R15
    62  0036  7C  00                   ADCI $00        ;ADD CARRY
    63  0038  BF                       PHI R15
    64  0039  28                DEC:   DEC R8
    65  003A  88                       GLO R8          ;TEST IF ALL 8 BITS DONE
    66  003B  3A  27                   BNZ LOOP        ;LOOP TILL DONE
    67  003D  8F                       GLO R15
    68  003E  73                       STXD            ;MOVE TO STACK
    69  003F  9F                       GHI R15
    70  0040  52                       STR R2          ;HIGH BYTE LAST
    71  0041  30  1D                   BR END          ;FINISHED
    72                                 ;
    73                                 ;THIS SUB ROUTINE WAITS
    74                                 ;FOR I PRESSED
    75                                 ;DEBOUNCES & RETURNS
    76  0043  D0                DONE:  SEP R0          ;RETURN TO CALLER
    77  0044  3F  44             HERE:  BN4 HERE        ;LOOP HERE TILL PRESSED
    78  0046  F8  06                   LDI $06         ;DEBOUNCE TIME
    79  0048  B8                       PHI R8
    80  0049  28                DELAY: DEC R8
    81  004A  98                       GHI R8
    82  004B  37  4B             LOW:   B4 LOW          ;LOOP HERE TILL RELEASED
    83  004D  3A  49                   BNZ DELAY       ;WASTE A FEW MILISECONDS
    84  004F  30  43                   BR DONE
    85                                 ;
    86        0000                     .END


.MAIN.  CROSS - MICRO PROCESSOR ASSEMBLER 6(31)  13-AUG-83  19:06  PAGE 1-2
END.M18 SYMBOL TABLE

DEC     0039            DELAY   0049            DONE    0043            END     001D
HERE    0044            LOOP    0027            LOW     004B            MUL     001E
R0    =%0000            R1    =%0001            R10   =%000A            R11   =%000B
R12   =%000C            R13   =%000D            R14   =%000E            R15   =%000F
R2    =%0002            R3    =%0003            R4    =%0004            R5    =%0005
R6    =%0006            R7    =%0007            R8    =%0008            R9    =%0009
START   0009            .     = 0051


 ERRORS DETECTED:  0

 *MPLY/PTP,MPLY=MPLY,END
 RUN-TIME:  0  0  0 SECONDS
 CORE USED:  3K
```

Steve Nies
Rt 1 #30 Iroquois Trail
Ruckersville, VA 22968
Phone: (804) 985-7074

AN 1802 ASSEMBLER - FORTH STYLE!

Having chosen FORTH as THE language for my 1802 system, I needed a method by which I could include machine code routines into FORTH definitions. Not wanting to reinvent the wheel, I decided to look for an already written assembler that I could include in my FORTH nucleus. However, I noticed that every FORTH 1802 assembler (that I am aware of) worries about page boundaries. Each assembler generates an error code if the target of a branch is not on the same page as the branch opcode. That's when I decided to design an assembler that would automatically decide whether a branch should be long or short, depending on the branch's target address.

The key to automatically selecting the type of the branch is to always store enough space for a two byte address, even though a one byte address may be enough. I know at first glance this method seems wasteful of memory, but the advantages of this method far outweigh the disadvantages. The main advantage to this method is that a program may be assembled at any location without regard to page boundaries. This advantage is absolutely necessary since FORTH doesn't take into account page boundaries when it is storing definitions into the dictionary. If this method was not used, a definition may or may not assemble, depending on the number and length of definitions previously compiled.

The only other argument against this method that I can think of concerns the execution speed of the code. In some applications where code speed is critical, a long branch may take too much time. In this case (and ONLY this case), the SHORT pseudo-op was created to force the assembler to use a short branch. This word should only be used when the assembly origin is known in advance and it is absolutely impossible to use a long branch.

The remainder of the program is just a typical FORTH assembler. Of course, labels are omitted in favor of structured constructs such as BEGIN ... UNTIL and IF ... ELSE ... ENDIF. Remember that the assembler uses reverse polish notation. For example, to load the low half of the address of ARRAY[3] into the accumulator, you would code { ARRAY 3 + LDI, }. Note that the braces are only used to delimit the FORTH text from the text of the article and should not be keyed into the definition.

---------- ASM1802F SYNTAX ----------

Before the syntax of the assembler is mentioned, I need to explain the distinctions between various symbols that I will be using. Following is a list of the symbols with a short explanation of each.

bytexpr = the result from any FORTH mathematical expression. Only the low byte of the result will be used. The result of the expression will be an unsigned number from 0 to 255 or a signed number from -128 to +127.

wordexpr = the result from any FORTH mathematical expression. The result of the expression will be an unsigned number from 0 to 65535 or a signed number from -32768 to +32767.

nibexpr = the result from any FORTH mathematical expression. The result of the expression must be positive and less than 16. Otherwise, an error message will be generated.

low3expr = the result from any FORTH mathematical expression. The result of the expression must be positive and less than 8. Otherwise, an error message will be generated.

..code.. = a sequence of 1802 assembly instructions.

condition = one of the status or EF flags to be tested. The conditions available to be tested are Q, Z, DF, EF1, EF2, EF3, and EF4.

The remainder of this article lists the various types of instructions and pseudo-ops available to the user. The program listing might also be helpful in answering any further questions.

One byte instructions with no operands:

IDL,	SKP,	RET,	DIS,
LDXA,	STXD,	ADC,	SDB,
SHRC,	SMB,	SAV,	MARK,
REQ,	SEQ,	SHLC,	NOP,
LSNQ,	LSNZ,	LSNF,	LSKP,
LSIE,	LSQ,	LSZ,	LSDF,
LDX,	OR,	AND,	XOR,
ADD,	SD,	SHR,	SM,
SHL,			

One byte instructions with one operand:

nibexpr LDN,	nibexpr INC,	nibexpr DEC,	nibexpr LDA,
nibexpr STR,	nibexpr GLO,	nibexpr GHI,	nibexpr PLO,
nibexpr PHI,	nibexpr SEP,	nibexpr SEX,	low3expr INP,
low3expr OUT,			

Two byte instructions:

byteexpr ADCI,	byteexpr SDBI,	byteexpr SMBI,	byteexpr LDI,
byteexpr ORI,	byteexpr ANI,	byteexpr XRI,	byteexpr ADI,
byteexpr SDI,	byteexpr SMI,		

Three byte instructions:

wordexpr LBR,

Pseudo-ops:

CODE = begins a definition coded in 1802 assembly language.

PAGE = move the dictionary pointer (DP) up to the next page. This pseudo-op is most often used with the SHORT pseudo-op to guarantee that an assembler definition will begin on a page boundary.

Pseudo-ops (continued):

SHORT = force the assembler to use a short branch. An error message will be generated if the target of the branch is not on the same page as the branch's opcode.

NOT = reverse the condition of a branch. For example, the sequence { 0 IF ..code.. } will execute the code following the IF statement if 0 is true. The sequence { 0 NOT IF ..code.. } will execute the code following the IF statement if 0 is false.

BEGIN, ..code.. condition UNTIL,
= execute the code following the BEGIN, until the condition is true.

BEGIN, ..code.. condition WHILE, ..code.. REPEAT,
= execute the code following the BEGIN, and WHILE, as long as the condition is true. Note that the code following the BEGIN, will be executed at least once.

condition IF, ..code.. ENDIF,
= execute the code following the IF, if the condition is true. Otherwise, jump to the ENDIF,.

condition IF, ..code.. ELSE, ..code.. ENDIF,
= execute the code following the IF, if the condition is true. Otherwise, execute the code following the ELSE,. In both cases, following execution of the appropriate code, execution will jump to the ENDIF,.

NEXT = terminate an assembly language definition.

```
================================================================
==                    ASM1802F program listing                  ==
================================================================

HEX
VOCABULARY ASSEMBLER IMMEDIATE
ASSEMBLER DEFINITIONS

32 USER BR_FLAG      ( 0=long & short branches, 1=short branches only )
: ERR
   ( ------------------------------------------------------------ )
   ( Print error message n                                        )
   ( Usage:  n ERR ---  where n = error message number            )
   ( ------------------------------------------------------------ )
   MESSAGE 0 BR_FLAG ! SP! QUIT ;

: ?ERR
   ( ------------------------------------------------------------ )
   ( Print error message n if flag is true                        )
   ( Usage:  f n ?ERR ---                                         )
   ( ------------------------------------------------------------ )
   SWAP IF ERR ELSE DROP ENDIF ;
```

```
: PAGE?
   ( ------------------------------------------------------------- )
   ( See if the target of the branch is on the same page as the DP  )
   ( Usage:  addr PAGE? addr f   where:                             )
   (            addr = (address of the branch's opcode) + 1         )
   (            f = true if the target is on the same page          )
   (            f = false if the target is not on the same page     )
   ( ------------------------------------------------------------- )
   DUP FF00 AND HERE FF00 AND = ;

: FLAG?
   ( ------------------------------------------------------------- )
   ( See if the branch is based on one of the EF flags              )
   ( Usage:  opcode FLAG? opcode f   where:                         )
   (            f = true if branch is based on EF1, EF2, EF3, or EF4 )
   (            f = false if branch is based on Q, Z, or DF         )
   ( ------------------------------------------------------------- )
   DUP 37 > OVER 3C < AND OVER 34 < OR 0= ;

: LBR, C0 C, , ;          ( This instruction is available to the user )

0706 VARIABLE OFT 0504 , 0303 , 03 C,
: BUMP?
   ( ------------------------------------------------------------- )
   ( See if space exists on the current page to assemble a          )
   (    conditional branch around a LBR.  The DP cannot be pointing )
   (    to the last 7 bytes on the page in order to have sufficient  )
   (    space to assemble the branches.  If the DP is pointing to one )
   (    of the last 7 bytes on a page, this routine will assemble a  )
   (    LBR to the next page and adjust the DP accordingly.         )
   ( Usage:  --- BUMP? ---                                          )
   ( ------------------------------------------------------------- )
   HERE FF AND F8 >              ( Is there room to jump around a LBR? )
   IF HERE FF AND F9 - OFT + C@ ( No, so determine size of the bump )
      HERE + DUP LBR, DP !       ( Assemble the LBR and bump the DP )
   ENDIF ;

: LEVEL!
   ( ------------------------------------------------------------- )
   ( Push the conditional pairing level on the stack.  If the SHORT  )
   (    pseudo-op was specified, set the SHORT flag contained in the )
   (    pairing level number.                                       )
   ( Usage:  n LEVEL! n   where: n = the conditional pairing level  )
   ( ------------------------------------------------------------- )
   BR_FLAG @                     ( Was the SHORT pseudo-op specified? )
   IF 1 OR                       ( Yes, so modify the pairing number )
   ENDIF
   0 BR_FLAG ! ;                 ( Reset the SHORT pseudo-op )

: @LEVEL
   ( ------------------------------------------------------------- )
   ( Mask off the SHORT flag contained in the conditional level     )
   (    number, then compare the two level numbers and generate an  )
   (    error number if not equal.                                  )
   ( Usage:  n1 n2 @LEVEL ---  where:                               )
   (            n1, n2 = level numbers to verify conditional pairing )
   ( ------------------------------------------------------------- )
   SWAP DUP 1 AND                ( Get SHORT bit from pairing number )
   BR_FLAG @ OR BR_FLAG !        ( Save state of the SHORT pseudo-op )
   FE AND = 0= 13 ?ERR ;         ( Are the two pairing numbers equal? )
```

```
: FORW_BR!
   ( ------------------------------------------------------------------ )
   ( Store the branch opcode and allocate space for the branch's       )
   (    address.  The target of the branch will be filled in later.    )
   ( Usage:  test FORW_BR! addr  where:                                )
   (            test = the condition to be tested                      )
   (            addr = (address of the branch's opcode) + 1            )
   ( ------------------------------------------------------------------ )
   C,                              ( Store the branch's opcode )
   HERE C4C4 BR_FLAG @             ( Was the SHORT pseudo-op specified? )
   IF C,                           ( Yes, store one byte branch address )
   ELSE ,                          ( No, store two byte branch address )
   ENDIF ;

: BACK_BR!
   ( ------------------------------------------------------------------ )
   ( Store a conditional branch to the address contained on the stack  )
   ( Usage:  addr test BACK_BR! ---  where:                            )
   (            test = the condition to be tested                      )
   (            addr = target address of the branch                    )
   ( ------------------------------------------------------------------ )
   C, PAGE?                        ( Store the branch opcode )
   IF C,                           ( Yes, so store the branch address )
   ELSE BR_FLAG @                  ( No, is a short branch mandatory? )
      IF 5 ERR                     ( Yes, so generate an error message )
      ELSE -1 ALLOT                ( No, move DP back to stored opcode )
         HERE C@ FLAG?             ( See if the opcode is an EF flag )
         IF BUMP?                  ( See if we have to bump the DP )
            SP@ 1+ 8 TOGGLE C,     ( Store a conditional jump around LBR )
            HERE 4 + C, LBR,       ( Assemble a LBR )
         ELSE 90 + C, ,            ( Make short branch into a long branch )
         ENDIF
      ENDIF
   ENDIF ;

: FIX_BR
   ( ------------------------------------------------------------------ )
   ( Fix the address of the last branch to point to the current        )
   (    location of the DP.  If the branch is on the same page as the  )
   (    DP, then use a short branch.  Otherwise, use a long branch.    )
   ( Usage:  addr FIX_BR ---   where:                                  )
   (            addr = (address of the branch's opcode) + 1            )
   ( ------------------------------------------------------------------ )
   PAGE?                           ( Is DP is on same page as branch? )
   IF HERE SWAP C!                 ( Yes, so use a short branch )
   ELSE BR_FLAG @                  ( No, was SHORT pseudo-op specified? )
      IF 5 ERR                     ( Yes, so generate an error )
      ELSE 1 - DUP C@ FLAG?        ( No, is the branch is based on EFx? )
         IF BUMP?                  ( It is, see if we have to bump DP )
            30 C, HERE 6 + C,      ( Branch around the kludge )
            C, HERE 4 + C,         ( Store a conditional jump around ... )
            DUP 3 + LBR,           ( ... LBR and then assemble a LBR ... )
                                   ( ... back to the original branch. )
            C0 OVER C!             ( Finally, replace original test ... )
            HERE 5 - SWAP 1+ !     ( ... with a LBR to the kludge )
         ELSE 90 + OVER C!         ( Use the long version of the opcode )
            HERE SWAP 1+ !         ( Store full addr after the branch )
         ENDIF
      ENDIF
   ENDIF ;
```

```
: LOW3 SWAP DUP 7 > OVER 1 < OR 5 ?ERR OR C, ;
: M.5 <BUILDS C, DOES> C@ SWAP DUP F > OVER 0< OR 5 ?ERR OR C, ;
: M1 <BUILDS C, DOES> C@ C, ;
: M2 <BUILDS C, DOES> C@ C, DUP ABS FF > 5 ?ERR C, ;

( ================================================================ )
(     The remainder of the listing contains the assembly language    )
(     mnemonics and pseudo-ops available to the user.                )
( ================================================================ )

   00 M1 IDL,      00 M.5 LDN,    10 M.5 INC,    20 M.5 DEC,
   38 M1 SKP,      40 M.5 LDA,    50 M.5 STR,    70 M1 RET,
   71 M1 DIS,      72 M1 LDXA,    73 M1 STXD,    74 M1 ADC,
   75 M1 SDB,      76 M1 SHRC,    77 M1 SMB,     78 M1 SAV,
   79 M1 MARK,     7A M1 REQ,     7B M1 SEQ,     7C M2 ADCI,
   7D M2 SDBI,     7E M1 SHLC,    7F M2 SMBI,    80 M.5 GLO,
   90 M.5 GHI,     A0 M.5 PLO,    B0 M.5 PHI,    C4 M1 NOP,
   C5 M1 LSNQ,     C6 M1 LSNZ,    C7 M1 LSNF,    C8 M1 LSKP,
   CC M1 LSIE,     CD M1 LSQ,     CE M1 LSZ,     CF M1 LSDF,
   D0 M.5 SEP,     E0 M.5 SEX,    F0 M1 LDX,     F1 M1 OR,
   F2 M1 AND,      F3 M1 XOR,     F4 M1 ADD,     F5 M1 SD,
   F6 M1 SHR,      F7 M1 SM,      F8 M2 LDI,     F9 M2 ORI,
   FA M2 ANI,      FB M2 XRI,     FC M2 ADI,     FD M2 SDI,
   FE M1 SHL,      FF M2 SMI,

: INP, 68 LOW3 ;
: OUT, 60 LOW3 ;

: CODE
   ( ------------------------------------------------------------- )
   ( Begins the assembly of 1802 mnemonics into a definition       )
   ( Usage:  --- CODE ---                                          )
   ( ------------------------------------------------------------- )
   CREATE [COMPILE] ASSEMBLER !CSP 0 BR_FLAG ! ; IMMEDIATE

: PAGE
   ( ------------------------------------------------------------- )
   ( Force the dictionary pointer to be moved to the start of the  )
   (    next 256 byte page boundary.  Using this pseudo-op guarantees )
   (    that an assembler definition will begin on a page boundary. )
   ( Usage:  --- PAGE ---                                          )
   ( ------------------------------------------------------------- )
   HERE FF00 AND 100 + DP ! ;

: SHORT
   ( ------------------------------------------------------------- )
   ( Force the assembler to use a short branch.  An error will be  )
   (    generated if the current location of the DP is not on the  )
   (    same page as the branch.  This pseudo-op should only be used )
   (    when the execution time of the code is critical.           )
   ( Usage:  --- SHORT ---   Note:  The SHORT psuedo-op is specified )
   (           by placing the word 'SHORT' between the condition to )
   (           be tested and the branch condition.  For example:    )
   (                .... DF SHORT IF, .....                        )
   ( ------------------------------------------------------------- )
   1 BR_FLAG ! ;
```

```
: NOT
   ( ------------------------------------------------------------ )
   ( Reverse the branch condition                                 )
   ( Usage:  opcode NOT opcode                                    )
   ( ------------------------------------------------------------ )
   SP@ 1+ 8 TOGGLE ;
       ( ... The following are the available branch conditions ... )
   : Q 39 ;    : Z 3A ;     : DF 3B ;
   : EF1 3C;   : EF2 3D ;   : EF3 3E ;   : EF4 3F ;

: BEGIN,
   ( ------------------------------------------------------------ )
   ( Start a BEGIN, ... UNTIL, loop or BEGIN, ... WHILE, ... REPEAT,  )
   (    loop.  This word saves the address of the following code for  )
   (    future use.                                               )
   ( Usage:  --- BEGIN, addr 8   where:                           )
   (           addr = the address of the code following the BEGIN,    )
   (           8 = an internal flag used for error checking       )
   ( ------------------------------------------------------------ )
   HERE 8 ;

: UNTIL,
   ( ------------------------------------------------------------ )
   ( Repeat the loop defined by BEGIN, .. UNTIL, until the specified  )
   (    test is true.                                             )
   ( Usage:  addr 8 test UNTIL, ---   where:                      )
   (           addr = the address of the code following the BEGIN,    )
   (           8 = an internal flag for error checking            )
   (           test = one of the flags to be tested (ie. Z, EFx, etc) )
   ( ------------------------------------------------------------ )
   SWAP 8 @LEVEL             ( See if the conditional levels match )
   BACK_BR! 0 BR_FLAG ! ;    ( Generate a branch to the BEGIN, )

: WHILE,
   ( ------------------------------------------------------------ )
   ( Create a loop whose boundaries are defined by the preceeding     )
   (    BEGIN, and the following REPEAT,.  The loop will be repeated  )
   (    as long as the specified test is true.                    )
   ( Usage:  addr1 8 test WHILE, addr1 addr2 0A  where:           )
   (           addr1 = the address of the code following the BEGIN,   )
   (           8, 0A = internal flags for error checking          )
   (           test = one of the flags to be tested (ie. Z, EFx, etc) )
   (           addr2 = (address of the branch opcode) + 1         )
   ( ------------------------------------------------------------ )
   SWAP 8 @LEVEL             ( See if the conditional levels match )
   FORW_BR!                  ( Store branch's opcode and address )
   A LEVEL! ;                ( Save the conditional pairing number )

: REPEAT,
   ( ------------------------------------------------------------ )
   ( Terminate the loop defined by the preceeding BEGIN, and WHILE,.  )
   (    The loop will repeat as long as the specified test is true    )
   ( Usage:  addr1 addr2 0A REPEAT, ---   where:                  )
   (           addr1 = the address of the code following the BEGIN,   )
   (           addr2 = (address of the branch opcode) + 1         )
   (           0A = an internal flag for error checking           )
   ( ------------------------------------------------------------ )
   A @LEVEL                  ( See if the conditional levels match )
   SWAP 30 BACK_BR!          ( Generate a branch to the BEGIN, )
   FIX_BR 0 BR_FLAG ! ;      ( Resolve forward branch from the WHILE, )
```

```
: IF,
   ( ------------------------------------------------------------- )
   ( Execute the following code if the test specified is true.  If  )
   (    the test is false, jump to the following ELSE,.  If ELSE, is )
   (    not specified, then jump to the following ENDIF,            )
   ( Usage:  test IF, addr 4   where:                               )
   (            test = one of the flags to test (ie. Z, EFx, etc)   )
   (            addr = (address of the branch opcode) + 1           )
   (            4 = an internal flag for error checking             )
   ( ------------------------------------------------------------- )
   FORW_BR!                  ( Store branch's opcode and address )
   4 LEVEL! ;                ( Save the conditional pairing number )

: ELSE,
   ( ------------------------------------------------------------- )
   ( Execute the following code if the test from the preceeding IF  )
   (    was false.                                                  )
   ( Usage:  addr1 4 ELSE, addr2 6   where:                         )
   (            addr1 = (address of the branch opcode) + 1          )
   (            4,6 = internal flags for error checking             )
   (            addr2 = the address of the unconditional jump       )
   (                    immediately preceeding the ELSE,.  The code )
   (                    that follows the IF uses this unconditional )
   (                    jump to skip over the code executed by ELSE, )
   ( ------------------------------------------------------------- )
   4 @LEVEL                  ( See if the conditional levels match )
   30 FORW_BR!               ( Store branch's opcode and address )
   SWAP FIX_BR               ( Now resolve forward branch from the IF, )
   6 LEVEL! ;                ( Save the conditional pairing number )

: ENDIF,
   ( ------------------------------------------------------------- )
   ( Terminate the most recent IF, or ELSE                          )
   ( Usage:  addr 4 ENDIF, ---                                      )
   (              OR                                                )
   (         addr 6 ENDIF, ---   where:                             )
   (            addr = (address of the most recent branch) + 1      )
   (            4, 6 = internal flags used for error checking       )
   ( ------------------------------------------------------------- )
   FD AND 4 @LEVEL           ( See if the conditional levels match )
   FIX_BR                    ( Resolve forward branch from IF, or ELSE, )
   0 BR_FLAG ! ;             ( Reset the SHORT pseudo-op )

: NEXT
   ( ------------------------------------------------------------- )
   ( Terminate the procedure coded in 1802 assembly                 )
   ( Usage:  --- NEXT ---                                           )
   ( ------------------------------------------------------------- )
   C SEP, CURRENT @ CONTEXT ! ?EXEC ?CSP SMUDGE ;
```

SOME HINTS ON PROGRAM DEBUGGING

-by Dick Thornton 1403 Mormac Road, Richmond, Va. 23229

When a program fails to work properly, it is first necessary to determine whether the failure is due to hardware, software, or a combination of the two. For this discussion, I am assuming that the fault is definitely in the program.

The first requirement in program debugging is to understand what the program is supposed to do, and how it works. If you wrote the program that is no problem, but if written by someone else, you will have to learn how it is supposed to work. With any program, it is always possible that it was not loaded properly, so a good place to start would be to verify that the program is correct by comparing the content of memory with a listing of the program. Be particularly careful of 8's and B's, which are often difficult to distinguish on listings.

After verifying that the program is indeed loaded correctly, it is time to gather information about the problem. Try everything the program is supposed to do, and make notes of what happens. At this point it is best not to be concerned with what may have caused a given problem. Just write down the test and its result and go on. Write down results of tests that produced correct results, too. When you have finished, use your list to determine which functions are performing correctly, and which are failing. Most programs of any size use routines performed via SCRT or SEP calls. If all functions using a particular routine fail, then the routine may be at fault. However, if other program functions that work also use the routine, it is less likely to be a problem. Spend some time with your analysis here, and you may be able to pinpoint the problem if you are a good detective. Remember that everything you see is a clue, and the program operates on strictly logical principles using the program instructions in memory.

One common problem with a new program is that it appears to do nothing at all, or produces screwy results before you have done anything at all. In this case, the program is failing during initialization. Another typical problem is that the program functions properly up to a certain point, then works improperly or not at all. The program may invariably perform some functions correctly, while others always produce incorrect results. The worst problem I've found is when a given function suddenly begins to fail after working properly for some time. In this case, some other function is often at fault, and is setting up the conditions which cause the failure. By trying different sequences of operation, you may be able to determine which function creates the problem.

The problem determination step should have given you some ideas about where the problem(s) may be, whether in initialization, the main processing routine, or one or more other routines. Now you

should consider the possible conditions which might cause the functions to fail. First, the routine itself may be correct, and the failure be caused by addresses, data, or register values which are incorrect on entry to the routine. If the conditions on entry are correct, there may be a logic problem within the routine. Simple logic problems involving incorrect decisions and manipulations usually result in only that one function being at fault, while other parts of the program work properly. If the routine changes its PC register, all sorts of crazy things can happen, as the change to the PC register is equivalent to a branch instruction. Improper use of the X-register can also cause wierd problems. Often, the X-register must be changed during processing, for use by one of the indexed instructions, or to address a data stack. If the X-register is not changed when it should have been, or is not reset when it should be, strange things can happen. Finally, a problem can occur at the end of a routine. This might be that data or addresses are not set correctly, which will later cause a problem for some other function. If the routine is performed by SCRT or SEP calls, it must return to the caller via a SEP instruction. If the SEP is for the wrong register, the program will go astray. Another problem when leaving a routine is the condition of the stack and the stack pointer register (usually R2). For SCRT calls, the stack must be left exactly as it was found on entry. If data on the stack is changed incorrectly, or if the stack pointer register does not contain the same address it had on entry to the routine, the SCRT return will get all balled up, and you will not return correctly. Fertile areas for investigation, then, are the conditions which exist on entry to the routine, and at exit from the routine. Check on entry that all registers and data required are correct. If not, you must backtrack to determine where they got messed up. The routine cannot be expected to perform properly if it doesn't have correct data to work with. Next, check the conditions at exit from the routine. Does the stack pointer have the same address it had at entry to the routine? Is the returning SEP to the correct register? Has stack data been changed incorrectly? Did the routine produce the desired result? Finally, did you ever get to the exit point?

Now that you know what to look for, let's see how to do it. If you have a monitor, this is not too difficult. If the monitor has a breakpoint facility like that used by Steve Nies' monitor, merely set breakpoints at the appropriate points, then list the values of the registers and appropriate data areas using monitor display functions when the breakpoint is encountered. Without a breakpoint facility, you must create your own breakpoint. I do this by placing 7B00 in the program where I want it to stop. This will turn on the Q lamp and stop processing. Now you need to understand that the 1802 sets X=0, P=0, R0=0000, Q=0, N-lines=000, and IE=1 when you switch to reset. The important thing here is that the contents of R1-RF, D, and DF are not changed when you reset the machine, and that means you can save their contents and look at them with a

monitor memory display. I am using a monitor that saves the contents of registers each time it is entered, and has a facility for displaying registers. If this is available to you, merely switch to RESET, then to RUN (if the long branch to your monitor is in locations 0000-0002) and display the registers. If your monitor does not have this facility, you must plan your testing more carefully, and provide a register save program before running tests. The 7B00-RESET-RUN method imposes certain restrictions on your program. First, the program should not use R0 as its PC, since the value of R0 is changed by RESET. At least one other register should be free of use by the program so that it can be used by the register save program. The program should not use the Q lamp, as it is used to tell you that you've encountered your breakpoint. You'll need 93 bytes of memory for the register save program and the register save area. If R0 is used as the program's PC, you will not be able to determine where the program was executing at the time you switched to RESET, and this can be very helpful. If the program uses all the registers, you must choose one for use by the register save program that is least likely to be of value in troubleshooting. Before starting your test, load the following register save program into an unused area of memory:

```
LOC   HEX        COMMENT
0100  F8FFA1     R1.0 = address of
0103  F801B1           the register save area
0106  E1         X=1
0107  8F739F73   save RF
010B  8E739E73   save RE
010F  8D739D73   save RD
0113  8C739C73   save RC
0117  8B739B73   save RB
011B  8A739A73   save RA
011F  89739973   save R9
0123  88739873   save R8
0127  87739773   save R7
012B  86739673   save R6
012F  85739573   save R5
0133  84739473   save R4
0137  83739373   save R3
013B  82739273   save R2
013F  7B00
```

I have shown the register save program to be loaded at locations 0100-0140, but it could be placed anywhere you have free memory. Also, I show the rightmost byte of the register save area to be at location 01FF (the first two lines load this address into R1), and this can also be any free area of memory. Finally, I show R1 used as the X register during register save. Any other register desired could be used instead. If another register is used as X, change the corresponding register save instruction to save R1, instead. As written, the register contents will be in memory as

follows: R2 at 01E4-01E5, R3 at 01E6-01E7, R4 at 01E8-01E9, R5 at 01EA-01EB, R6 at 01EC-01ED, R7 at 01EE-01EF, R8 at 01F0-01F1, R9 at 01F2-01F3, RA at 01F4-01F5, RB at 01F6-01F7, RC at 01F8-01F9, RD at 01FA-01FB, RE at 01FC-01FD, and RF at 01FE-01FF. If you use another location for the save area, write down the addresses of the registers where you've saved them.

To run your test, first load the register save program and the program to be tested into memory. Using your monitor memory change function, put the 7B00 into the program at the point you've identified as worth investigation. Now execute the program. One of two things will happen: the Q lamp will come on, or it won't. If it comes on, you've gotten to the breakpoint, if not, you never got there. In either event, switch to RESET, then to LOAD and key in a long branch to your register save routine. If the routine was loaded at 0100, as shown above, enter C00100. Now switch to RESET, then to RUN. The Q lamp should come on, showing that the registers have been saved. Now switch to RESET, then to LOAD and put the long branch to your monitor back into locations 0000-0002. Switch to RESET, then to RUN to get to your monitor, and use its memory display function to examine the registers and any other memory appropriate to the problem. If you didn't get to your breakpoint, look at the content of the register used as PC by the program. The address in this register will tell you where you were executing at the time you switched to RESET, and should be an additional clue as to where the problem lies. Write down the content of all the registers and verify that all of them used contain valid addresses and data. If SCRT is used, locate the stack by the address in R2 and write down the data it contains. Try to determine if the information in the stack is correct.

If the problem is not solved at this point, start all over, using the next location you want to use for the 7B00. A couple of trips through the procedure should help you to locate the problem.

While the procedure shown here is primitive, it has helped me to locate and correct problems in every program I've written. When working with a program written by someone else, especially when you don't have a good, well documented source listing, you may have to consult with the author concerning your problems. He will be much better able to help if you supply a detailed description of the problems. When writing, also mention your hardware configuration, explaining what kinds of peripheral devices you are using, what port addresses they use, and whether they are serial or parallel.

When writing your own programs, consider that you may want to debug them, and leave locations 0000-0002 free for a long branch to the monitor or to the register save program. Don't use register 0 for the PC, and try to leave some other register free for the save program.

Thomas E. Jones
Berlinerstr. 20
D6944 W. Germany

(or) 295th Avn CO., SFTS Det.
APO N.Y. 09028

FIR-FORTH Expansion

The following expansions of Fig Forth are designed for systems running Fig Forth under the TEMON (Neis Monitor/T.E./UPII) operating system, but I believe the "TCALL" word may be of general interest. It allows you to call almost any SCRT machine language routine from within a Forth definition. Only one location must be changed to relocate it.

For all those who recieved a cassette, the manual, and listing of TEMON3.2, the Fig Forth additions will allow you to use many features of the operating system while still running Forth. The device independant nature of TEMON made this much easier to accomplish. Check the screen printer.

The new commands are:

```
CSAVE     ---    ( Save the current screen's disk block on cassette.)
CLOAD     ---    ( Load to the current screen's block a named cass file)
CVERIFY   ---    ( Verify a named cassette file against the current screen)
PRSCR     ---    ( Print the current screen on system printer )
```

Useful words added to the vocabulary include:

```
QON?      ---0/1 ( Test the Q flag )
ADDRESS   ---addr addr ( leave start and ending address of current screen's
                         block on the stack, start addr on top )
TKO       ---  ( xfer control to the TEMON O.S. and return )
TCALL data addr --- data  ( execute an SCRT routine, passing 16 bits to it
                            in RF and leaving the contents of RF on Forth
                            stack on return )
ASSIGN  device ch#---     ( Assign a named system peripheral to a logical
                            channel )
```

You will likely have to use your own printer motor on/off routines. Note also the patch to TEMON3.3 on the bottom of screen #2. The initial shipments of TEMON were rev 3.2, but everyone should by now have recieved the update to 3.3 which added George Mussers excellent improvements to the full screen text-editor. After patching rev. 3.3 change location D08D to 34 to make the header print TEMON3.4 on turnon.

To install TCALL the scenario runs like this;

```
HEX HERE U. 1D16  OK
CREATE TCALL E2 C, SMUDGE  OK
' TCALL U. 1D2A OK
DECIMAL 76 ALLOT  OK
HEX  HERE U. 1D6D  OK
FORTH DEFINITIONS DECIMAL OK
LATEST 12 +ORIGIN
HERE 28 +ORIGIN
HERE 30 +ORIGIN
HERE FENCE !
BYE
MONITOR
> L 'TCALL.BIN'   (Or use Memory Exam to key it in at 1D2A)
```

Of course your HERE will probably be different from mine, and don't forget to relocate location 1D52 in the TCALL listings.

```
                        #
0000                    ;.........................32
0000                    ;     TCALL                .        FORTH 1.2
0000                    ;.........................
0000                    ;   TOM JONES 6/83
0000                    ;
0000                    ;SCRT CALL FOR FIG-FORTH
0000                    ;EXPECTS 2 DATA BYTES AND TWO  ADDRESS BYTES ON FORTH I-STK,
0000                    ; HI DAT LO DAT HI ADR LO ADR --- RF.0 RF.1
0000                    ;RETURNS RF ON FORTH STACK.
0000                    ;TO LOCATE ORIGIN TYPE:
0000                    ;     CREATE TCALL DC C, SMUDGE OK     ' TCALL . 1D2A  OK
0000                    ;
0000                        ORG #1D2A
1D2A  E2                TCALL:  SEX R2
1D2B  9A                        GHI RA
1D2C  73                        STXD
1D2D  8A                        GLO RA
1D2E  73                        STXD
1D2F  9B                        GHI RB
1D30  73                        STXD
1D31  8B                        GLO RB
1D32  73                        STXD
1D33  9C                        GHI RC
1D34  73                        STXD
1D35  8C                        GLO RC
1D36  73                        STXD
1D37  9D                        GHI RD
1D38  73                        STXD
1D39  8D                        GLO RD
1D3A  73                        STXD
1D3B  49                        LDA R9; ADDR -> R6
1D3C  B6                        PHI R6
1D3D  09                        LDN R9
1D3E  A6                        PLO R6
1D3F  29                        DEC R9
1D40  29                        DEC R9
1D41  09                        LDN R9; DATA -> RF
1D42  AF                        PLO RF
1D43  29                        DEC R9
1D44  09                        LDN R9
1D45  BF                        PHI RF
1D46  29                        DEC R9; FIX I-STK
1D47  29                        DEC R9
1D48  99                        GHI R9; SAVE I-STK PTR
1D49  73                        STXD
1D4A  89                        GLO R9
1D4B  73                        STXD
1D4C  F800                      LDI #00 ;DUMMY RTN ADR
1D4E  73                        STXD
1D4F  73                        STXD
1D50  93                        GHI R3 ;HI ADR OF TRET
1D51  73                        STXD
1D52  F857                      LDI TRET
1D54  73                        STXD
1D55  9F                        GHI RF; "D"->RF.1
1D56  D5                        RETN; FROM FAKE CALL
1D57                    ;
1D57  12                TRET:   INC R2; RESTORE REGS
1D58  72                        LDXA
1D59  A9                        PLO R9
```

```
1D5A  72          LDXA
1D5B  B9          PHI R9
1D5C  72          LDXA
1D5D  AD          PLO RD
1D5E  72          LDXA
1D5F  BD          PHI RD
1D60  72          LDXA
1D61  AC          PLO RC
1D62  72          LDXA
1D63  BC          PHI RC
1D64  72          LDXA
1D65  AB          PLO RB
1D66  72          LDXA
1D67  BB          PHI RB
1D68  72          LDXA
1D69  AA          PLO RA
1D6A  02          LDN R2
1D6B  BA          PHI RA
1D6C  19          INC R9; SAVE I-STK REG
1D6D  19          INC R9
1D6E  9F          GHI RF
1D6F  59          STR R9
1D70  19          INC R9
1D71  8F          GLO RF
1D72  59          STR R9
1D73  29          DEC R9
1D74  E9          SEX R9
1D75  DC          SEP RC; RETN TO INNER INTERPRETER.
1D76
```

I would like to pass on to you two ways to make Fig Forth go into the "CRASH AND BURN" mode fast that you may have missed so far. One is to take Leo Brodies' book Starting FORTH too literally. On page 209 it states that the word CREATE puts the address of the code field when the created word is named. Not in our version, we execute the code immediately. To use the matrix creation method on page 207 you must get the matrix's address by using ' LIMIT with Fig Forth. Of course we also have to use SMUDGE.

Another good way to get strange stuff is to forget EMPTY-BUFFERS before loading a screen. If, for instance, you have something in SCR# 3 and use your monitor to load something else off tape to the block of ram-disk at 3400-37ff (on a 16K system), you will list the old contents with editor until EMPTY-BUFFERS is executed. The CLOAD command takes care of that for you automatically and lists the new file if it loaded error free.

If you don't have a copy of TEMON3.3 yet I still have some manuals and cassettes for $18. a copy.

```
SCR # 1
  Ø ( CSAVE/CLOAD          SCR1                6-JUL-83  TEJ )
  1 ( READS/WRITES CURRENT SCREEN'S RAM DISK BLOCK TO TAPE )
  2 HEX VARIABLE KBUFR ( TEMON INPUT BUFFER)
  3 D6F1 CONSTANT COMRCV ( TEMON COMMAND STRING INTERPRETER )
  4 CREATE SETSA 53 C, 41 C, 20 C, SMUDGE ( SA  )
  5 CREATE SETLD 4C C, 20 C, SMUDGE ( L )
  6 CREATE QON? 19 C, 19 C, 19 C, F8 C, ØØ C, C5 C, F8 C, Ø1 C,
  7 59 C, 29 C, F8 C, ØØ C, 59 C, DC C, SMUDGE
  8 : BINIT [COMPILE] HEX F03F F000 DO 20 I C! LOOP F000 KBUFR !
  9 EMPTY-BUFFERS ; ( INITIALIZE BUFFERS IN TEMON AND FORTH )
 10 : SINIT BINIT ' SETSA KBUFR @ 3 CMOVE 3 KBUFR +! ; ( SETUP )
 11 : ADDRESS SCR @ 400 * LO + DUP 3FF ROT + SWAP ;
 12 : TKO Ø COMRCV TCALL DROP ; ( PASS CONTROL TO TEMON )
 13 : AD2BUF Ø <# 20 HOLD #S #> KBUFR @ ROT SWAP ROT  CMOVE
 14    5 KBUFR +! ; ( ADR->BUFR IN ASCII)
 15 : FILE? ." ENTER FILENAME' (W/O LEADING APOST.) " CR ;     -->

SCR # 2
  Ø ( CSAVE/CLOAD            SCR2                  8-JUL-83 TEJ)
  1 : TWORD 27 KBUFR @ C! I   KBUFR +! F03F KBUFR @ DO KEY DUP I C!
  2  I KBUFR +! DUP EMIT 27 = IF LEAVE THEN LOOP ; ( GET FILENAME)
  3 : LINIT BINIT ' SETLD KBUFR @ 2 CMOVE 2 KBUFR +! ;
  4 : FNAME CR FILE? TWORD ; ( GET THE FILENAME )
  5 : CSAVE SINIT ADDRESS AD2BUF AD2BUF ( INITIALIZE CSI BUFFER )
  6   FNAME CR ." RECORDING" TKO CR ;
  7
  8 : CLOAD  LINIT FNAME ( PUT FILENAME TO CSI BUFR )
  9        ADDRESS  AD2BUF DROP     ( START ADDR TO CSI BUFR )
 10    CR ." LOADING FILE" CR TKO CR QON? IF ." TAPE ERROR" ELSE
 11    [COMPILE] EDITOR SCR @ LIST THEN ; ( LOAD A FILE COMMAND)
 12
 13 ( PATCH TEMON3.3  D6F1 - F8 F0 B9 D4 C1 2E C1 70 D5 )
 14 ( TCALL MUST BE PART OF THE FORTH VOCABULARY ALSO )
 15 -->

SCR # 3
  Ø ( CVERIFY AND OTHER TEMON COMMANDS )
  1 CREATE SETV 56 C, 20 C, SMUDGE  ( VERIFY CMMD )
  2 : VINIT BINIT ' SETV KBUFR @ 2 CMOVE 2 KBUFR +! ;
  3 : CVERIFY VINIT FNAME ADDRESS AD2BUF DROP CR ." SEARCHING" CR
  4 TKO CR ; ( WILL VERIFY AGAINST THE CURRENT SCREEN, NOTE! )
  5 F0A1 CONSTANT CHAN ( LCT MATRIX START ADDRESS )
  6 C766 CONSTANT TV ( CONSOLE ) D1EA CONSTANT PRINTER ( SERIAL OUT)
  7 : ASSIGN 3 * CHAN + ! ;
  8
  9 : MTR-ON 1B EMIT 48 EMIT ;
 10 : MTR-OFF 1B EMIT 4A EMIT ;  ( CTLS, FOR TN300 PRINTER )
 11 : PRSCR PRINTER 8 ASSIGN MTR-ON SCR @ LIST MTR-OFF TV 8 ASSIGN ;
 12 ( PRINTS CURRENT SCREEN ONLY ON SERIAL PRINTER IN EDITOR MODE)
 13
 14
 15
```

Forth and the Smarterm-80

-By Michael Smith, 1 Cranleigh Court, Islington, Ontario, M9A 3Y2, Canada

Some time ago while I was tinkering with Forth, I found that there were a few things missing in Forth for my video board. Of course, if you don't have a Netronics Smarterm-80 these "problems" might not arise. The problem, or actually "problems" are those of cursor control, Forth commands being in upper case, and the graphic modes.

The first thing I wanted to change were the serial I/O routines that I was using. There was nothing really wrong with them, but a few changes would enormously increase the ease in which one could program. For one thing, I detest shift locks. Sure, they have their advantages—one doesn't have to press the shift key for the grueling four hours it takes to type in Tiny Pascal—but they also have their draw backs. Typing in alphabetics is fine, but what happens when one needs a number? You hit the four and a dollar sign appears on your CRT. Frustrated, the backspace is tapped and an "H" is now beside what should have been a four. Seeing the cause of the problem, the shift lock is unshifted and you backspace to the dollar sign. Two numbers are quickly entered followed by a stream of stack manipulation commands. After the carriage return is hit, Forth returns with "MSG #0" because it didn't understand the lower case commands following the two numbers....

A simple solution to this problem is to design a routine to convert all lower case ASCII into upper case. All this involves is checking if the data entered is between hex 61 and 7A inclusive (lower case "a" and "z" respectively). If the case is true, simply subtract hex 20 from the ASCII data and it will be transformed into an upper case character. Because of the check, all control and non-lower case characters are ignored.

That solves one of our problems simply enough, but what does one do about the graphic modes? The graphic modes are useful but they do have their associated problems. If a Forth screen is listed which has different graphic modes present, you are liable to see blinking sentences, funny Space Invader sort of characters, and other symbols and shapes which generally confuse more than help when you are debugging a Forth word.

To enter one of the 127 graphic modes on the Smarterm-80, a three letter sequence is used. Similarily, if one downshifts the keyboard to display special characters such as arrows, a three letter sequence would also be used. The only difference between the two sequences is the middle character. It seems then, that the solution would be just to change the middle character of all the graphic sequences to the character for the downshift sequence.

Unfortunately, the problem to this approach is twofold. The major problem that appears is that a large amount of time would be spent searching through the entered data, weeding out the graphic sequences and making the conversion, and then when the word with a graphic sequence in it is executed, all the downshifts have to be reconverted back into graphic sequences. If this reconversion is omitted, the whole purpose of the program is defeated. This brings us to the second problem. How does the program know which of the downshift sequences which it finds during the search prior to executing a word are actual downshifs or are converted graphic sequences?

A table could be used to keep track of all the conversions, but this would use a lot of memory if you had a graphics oriented program. All of this plus the actual program to go through the table of entries and convert the downshift sequence at that adress to a graphic sequence is more than my memory miser self would allow.

The solution which I settled on discards the three letter sequence and uses

a single character instead. This single character is not one of the standard ASCII codes. Rather than using a 7 bit ASCII code, I now use an 8 bit code. If the MSB is high, it means that the low 7 bits are the graphic mode number, which is the last character in the three letter sequence.

Normally, the graphic modes are entered by typing -ESCAPE- followed by a "G" (ie. a two character literal). Following this is the ASCII character which corresponds with the desired graphic mode number. With the 8 bit code, when the character output routine (the Forth word EMIT) sees that the MSB is high, it automatically outputs "-ESCAPE-G" (two characters) followed by the low seven bits of the original 8 bit character.

When one is entering a program, a simple poke makes the automatic printout into "-ESCAPE-Z" (the first two characters of the downshift sequence) instead of the graphic sequence. Once the program is ready to be executed, another poke can easily re-establish the graphic sequence.

Before I continue on to cursor control, I think I had better explain the actual use of the serial input routine. To enter a graphic mode from the keyboard, the input routine checks if you have pressed control underline (ASCII code 1F). If you have, it gets another character from the keyboard and sets the MSB high. This second character which it gets is equivalent to the third character one would use in the standard graphic sequence. Finally, this second character with the MSB set is the character that is sent to the Forth word KEY.

Another item which I have added to the input routine is the possibility of avoiding the lower to upper case conversion. Somewhere in memory which you designate, is a flag. If this flag is false, the conversion routine will be bypassed allowing one to enter lower case data. To change the status of this flag, the control C key is used. If the user wishes, he can change a byte so that a quotation mark will change the value of the flag allowing one to type all upper case outside of a quote and lower and upper case between two quotation marks.

Finally, we come to cursor control. The definition VHTAB is an extremely simple but useful word. Instead of moving the cursor around via spaces and linefeeds et cetera, one just has to enter the vertical co-ordinate and the horizontal co-ordinate. VHTAB will add an offset to both of the co-ordinates and then will reposition the cursor via absolute cursor addressing.

```
Example 1)   1 1 VHTAB 42 EMIT
        2)   12 40 VHTAB 42 EMIT
```

Example one will place an asterisk in the top left hand corner of the screen and example two will place it in the middle of the screen.

I hope with the additions mentioned above, programming in Forth with your Smarterm-80 will be simplified. Of course, if you don't have a Smarterm-80 the graphic and downshift sequences will be meaningless to you, but the lower to upper case conversion and the VHTAB definition may still be of some use.

So the next time you jab at your shift lock and it breaks or your graphics program is illegiable because it is made out of blinking blocks and half intensity vertical lines, don't fret about it. Enter the following three routines into your version of Forth and forget about having to unshift a shift lock ever again!

Input Routine

```
0000  9F            GHI RF        Get the entered character.
0001  59            STR R9        Store character on computation stack.
0002  29            DEC R9        Move stack pointer to high byte.
0003  F8 00         LDI #00
0005  59            STR R9        Set high byte to zero.
0006  60            IRX           Move stack pointer up one.
0007  72            LDXA
0008  B8            PHI R8        Restore high 8 bits of register 8.
0009  02            LDN R2
000A  A8            PLO R8        Restore low 8 bits of register 8.
000B  DC            SEP RC        Return to Forth.
000C  :  ENTER INPUT ROUTINE HERE.
000C  E2            SEX R2        Set R2 as stack pointer.
000D  19            INC R9
000E  19            INC R9
000F  19            INC R9        Move R9 to point to next location of computation
                                  stack.
0010  88            GLO R8
0011  73            STXD          Store low 8 bits of register 8 on stack.
0012  98            GHI R8
0013  73            STXD          Store high 8 bits of register 8 on stack.
0014  F8 XX         LDI #XX       Load low 8 bits of flag's address.
0016  A8            PLO R8
0017  F8 XX         LDI #XX       Load high 8 bits of flag's address.
                                  This address can be any free byte in memory.
0019  B8            PHI R8
001A  D4 XX YY      SEP #XXYY     SCRT call to character input routine.  Should
                                  return character in D and RF.1
001D  FF 03         SMI #03       Subtract value of control C from character.
001F  3A 29         BNZ #29       Continue checking for special keys if not
                                  control C.
0021  F8 01         LDI #01       Need a high LSB to toggle flag.
0023  E8            SEX R8        Set X equal to register pointing to flag.
0024  F3            XOR           Toggle flag.
0025  58            STR R8        Store new value of flag.
0026  E2            SEX R2        Restore value of X.
0027  30 1A         BR  #1A       Get another character from the keyboard.
0029  FF 1C         SMI #1C
002B  3A 34         BNZ #34       Continue checking if character is not control
                                  underline.
002D  D4 XX YY      SEP #XXYY     See address 001A for details.
0030  FC 80         ADI #80       Set MSB high.
0032  30 01         BR  #01       Store character and exit routine.
0034  08            LDN R8        Fetch flag's status.
0035  FA 01         ANI #01       Get rid of everything but the LSB.
0037  32 00         BZ  #00       Exit routine if flag false.
0039  9F            GHI RF        Fetch entered character.
003A  FF 61         SMI #61
003C  3B 00         BNF #00       Exit routine if character less than hex 61.
003E  FF 1A         SMI #1A       There are 26 letters in the alphabet.
0040  33 00         BDF #00       Exit routine if character greater or equal to
                                  hex 7B (61 plus 1A).
0042  FC 5B         ADI #5B       Convert lower case to upper case.
0044  30 01         BR  #01       Store converted character and exit routine.
```

Output Routine

```
0000  19          INC R9
0001  09          LDN R9        Fetch character to be printed from computation
                                stack.
0002  FE          SHL           Push the MSB into DF.
0003  3B 11       BNF #11       If the DF is low, that means that the character
                                is not a graphic sequence, hence print it directly.
0005  AF          PLO RF        Store the shifted character.  Note:  the contents
                                of RF.0 are destroyed.
0006  F8 1B       LDI #1B       Load the value of ASCII Escape.
0008  D4 XX YY    SEP #XXYY     Print the contents of D.
000B  F8 5A       LDI #5A       Load the value of ASCII Z.  Note:  If hex 47
                                is used instead of 5A, then this routine will
                                print graphics instead of downshifts.  Therefore,
                                to change from graphics to downshifts, just
                                change the value of memory location 000C.
000D  D4 XX YY    SEP #XXYY     Print the contents of D.
0010  8F          GLO RF        Get the shifted character from the beginning.
0011  F6          SHR           By shifting right, we set the MSB to zero.
0012  D4 XX YY    SEP #XXYY     Print the original character.
0015  29          DEC R9
0016  29          DEC R9
0017  29          DEC R9        Move the computation stack pointer to previous
                                entry.
0018  DC          SEP RC        Exit to Forth.
```

Important! Don't forget to put the address of the character output routine in memory location 0543/4 and the address of the character input routine in memory location 055E/F.

Cursor Control

```
FORTH DEFINITIONS HEX
CREATE VHTAB
F81B , ( Escape ) D4XX , YY C, ( Fill in the address of the character output routine )
F83D , ( Equal )  D4XX , YY C, ( SCRT call to character output routine )
2909 , ( Get vertical co-ordinate ) FC1F , ( Add offset )
D4XX , YY C, ( SCRT call to character output routine )
1919 , ( Increment R9 to point to top entry of computation stack )
09 C, ( Get the horizontal co-ordinate ) FC1F , ( Add offset )
D4XX , YY C, ( SCRT call to character output routine )
2929 , 29DC , ( Tidy up computation stack and leave machine language )
SMUDGE
```

1802 Computer Products Available from John Ware
- J. Ware, 2257 - 6th Ave., Fort Worth Texas, 76110

1. A 16k ram card that uses either 2114L or the cmos equivalent 1k by 4 rams. This can be expanded 1k at a time up to 16k maximum.

2. A 16k prom card that uses the 2716 5V chips. This card uses from one to 8 of the 2716 proms. Each has a separate address decoder and acts like 8 independent 2k cards. This allows you to set each prom address independently.

   Both cards are double sided with the holes plated through. Each is the same size as the Netronics cards with the same edge contact call out. The edge connectors are gold plated. Both are completely compatible with all existing Netronics cards now available for the ELF II. They come with complete instructions on assembly and use. The price is $35.00 each delivered by first class mail.

3. A 1.25k machine language terminal based monitor. This program comes with full instructions for use. The price is $2.00.

4. A much improved version of the above program on tape. This is 1.5k in length. Included are routines for both 300 and 2400 baud rate terminals. Provisions have been made to allow for different line length terminals and for custom I/O routines. The program is not in relocatable code but I have provided a short relocating program on the tape to make all of the necessary changes in the main program. This comes on a tape with a manual that includes an almost free printer interface. The price is $15.00.

   Either of the monitors will run out of prom without any changes other than the necessary address corrections. Both have tape routines that use the Netronics tape format. The second one has more commands and better input error checking. If you do much machine language programming the second one can pay for itself in two hours of use. Neither one uses the hex keypad in any way.

The above prices include delivery by first class mail anywhere in the USA. Outside of the US you will need to pay any taxes or other fees plus the additional postage. I will accept personal checks, cash, or money orders. On personal checks I wait until they clear my bank before shipping. In the event that I am temporarily out of something I hold your check until more are made. You will be informed immediatly if this happens. You may make the choice of getting your money back or waiting for the item.

Sincerely,

John Ware

John Ware
2257 6th Ave.
Ft Worth, Tx.
76110

Please complete this form and return. This information is desired to allow me to develop better hardware and software. This information will not be released to anyone else without your consent.

John Ware
2257 6th. Ave.
Ft. Worth,
Tx. 76110

Name_____
Address_____
City_____State_____Zip_____
Memory size_____Clock freq._____
What are ports used for?

Input port #	Output port #
1_____	1_____
2_____	2_____
3_____	3_____
4_____	4_____
5_____	5_____
6_____	6_____
7_____	7_____

What are flag lines used for?
1_____
2_____
3_____
4_____
Any other nonstandard things used
_____
_____
_____
_____
_____
_____
_____
_____
What other devices like printers, terminals, disk drives?
(Please include make and model and if terminal, baud rate, half/full duplex, and if parity used.)
_____
_____
_____
_____
_____
_____
_____
What hardware and software would you like to be able to have?
_____
_____
_____
_____
_____
_____
_____
Any other comments or suggestions?
_____
_____
_____
_____

NAME:_____ DATE:_____

PRODUCT ORDER	QUANTITY	UNIT PRICE	TOTAL
CPU Board	_____	$40.00	_____
Backplane and I/O Board, Ver. 2	_____	40.00	_____
Front Panel (with EPROM Burner, Clock)	_____	35.00	_____
VDU Board, Ver. 2	_____	40.00	_____
64K Dynamic (4116) Board	_____	50.00	_____
Netronics - Ace Adapter Board	_____	25.00	_____
I/O Adapter for Backplane, Ver. 1	_____	20.00	_____

Software

Fig FORTH - Netronics Cassette format (6K) 0000H	_____	$10.00	_____
Tiny Pilot - Netronics Cassette format (2K) 0000H	_____	$10.00	_____
SYMON - Netronics Cassette format (2K) C000H	_____	$10.00	_____

Back Issues

"Defacto" Year 1 - 3 (Edited)	_____	$20.00	_____
Year 4 Reprint	_____	10.00	_____
Year 5 Reprint	_____	10.00	_____
Year 6 Reprint	_____	10.00	_____

Membership - Year 7

Current Year - Sept. '83 - Aug. '84
includes 6 issues of Ipso Facto

Canadian	_____	$20.00 Cdn.	_____
American	_____	20.00 U.S.	_____
Overseas	_____	25.00 U.S.	_____

PRICE NOTE

Prices listed are in local funds. Americans and Overseas pay in U.S. Funds, Canadians in Canadian Funds. Overseas orders: for all items add $10.00 for air mail postage. Please use money orders or bank draft for prompt shipment. Personal cheques require up to six weeks for bank clearance prior to shipping orders.

SALE POLICY

We guarantee that all our products work in an A.C.E. configuration microcomputer. We will endeavour to assist in custom applications, but assume no liability for such use. Orders will be shipped as promptly as payment is guaranteed.

NAME: _____

MAILING ADDRESS: _____

_____

_____

_____

PHONE NO.: _____

Note: Ensure mailing address is correct, complete and printed.
Please ensure payment is enclosed.

ASSOCIATION OF COMPUTER-CHIP EXPERIMENTERS
c/o M.E. FRANKLIN
690 LAURIER AVENUE,
MILTON, ONTARIO
L9T 4R5
