

# Ipsos Facto

The A-C-E Magazine

JUNE 1981

ISSUE 23

## DEVOTED ENTIRELY TO THE COSMAC 1802

### ARTICLES

THE TEXT EDITOR	by Steve Nies	7
MODIFICATIONS TO THE CLUB VDU BOARD	by P. Muir	27
STATUS DISPLAY	by P. Muir	30
NETRONICS TINY BASIC FOR ALL 1802 SYSTEMS		
	by H.B. Stuurman	33
ELF II SERIAL I/O PACKAGE	by Wes Steiner	37
EPROM PROGRAMMING WITH AN 1802	by Ken Mantei	43

### DEPARTMENTS

1981 Executive	2
Publishers Page	2
Editorial Listing	3
Advertising Policy	3
Executive File	3-5
Help	5
Letters of Contact	5
Errata IPSO #20	5
For Sale or Swap	6
Club Order Form	49
Vote For Best Artical	49
Membership Form	49
Mailer	50

IPSO FACTO is published by the ASSOCIATION OF COMPUTER EXPERIMENTERS (A.C.E.), a non-profit, educational organization. Information in IPSO FACTO is believed to be accurate and reliable. However, no responsibility is assumed by IPSO FACTO or the ASSOCIATION OF COMPUTER EXPERIMENTERS for its use; nor for any infringements of patents or other rights of third parties which may result from its use.

ASSOCIATION OF COMPUTER EXPERIMENTERS 1981 EXECUTIVE

<u>President:</u> John Norris	416-239-8567	<u>Program Co-ordinator:</u> Jeff Davis	416-643-1578
<u>Past President:</u> Ken Bevis	416-277-2495	<u>Training Co-ordinators:</u> Fred Feaver	416-637-2513
<u>Secretary/Treasurer:</u> Mike Franklin	416-878-0740	Ken Bevis	
<u>Hardware Co-ordinator:</u> Anthony Tekatch	416-957-7556	<u>Software Co-Ordinator:</u> Wayne Bowdish	416-388-7116
<u>Hardware Production and Sales:</u> Fred Pluthero	416-389-4070	<u>Editor:</u> Fred Pluthero	416-389-4070
<u>Publishing Committee:</u> Dennis Mildon	416-385-0798	<u>Editorial Staff:</u> Sharon Swindells	
John Hanson	416-637-1076	<u>Consultant:</u> Bc. Silcox	416-681-2848
<u>Membership Co-ordinator:</u> Bernie Murphy	416-845-1630	<u>Draughtsman:</u> John Myszkowski	416-529-0250
Don MacKenzie	416-676-9084		

ARTICLE SUBMISSIONS:

We can always use lots of software and hardware related articles of all types. Inasmuch as editing consists of taking the path of least resistance, 'camera ready' articles stand the best chance of getting in. Camera ready means typed, single spaced, reasonably error free and done with a dark ribbon. Diagrams should be large and clear (we can reduce them) and clearly labelled. Don't let camera ready scare you off. If you don't have access to a typewriter, by all means send in what you have, we still want to see what you've been up to.

Some important notes: First, please send us your original manuscript, not a photocopy. The quality of most photocopies is invariably poor and such articles get pushed to the back of the editorial 'stack'. Second, make sure your diagrams and programs are accurate. We have enough trouble with errors on our part; there's no way we'll ever catch yours.

MEMBERSHIP RENEWALS:

This is a bit pointless; if you got this issue, you're fully paid up. To help you keep track, a note designating the status of your subscription should appear on the mailing label of your issue. This will most probably take the form of '2 of 6'.

US MEMBERS:

We understand that a number of our US members are having trouble obtaining Canadian currency. Don't bother, send in your membership renewal (or order for DeFacto, hint, hint) in US funds. This also applies to overseas members, if you can't come up with Canadian money easily enough, by all means send cheque or money order in US funds.

SEND ALL A.C.E. CORRESPONDENCE TO:

Bernie Murphy,  
102 McCraney Street,  
Oakville, Ontario. L6H 1H6  
Canada.

## EDITORIAL LISTING

Congratulations to D. Shroyer for his artical in IPSO FACTO 21 FULL BASIC FOR THE 1802 (REALLY!) It was the best artical. For this he receives a free membership in ACE for one year.

In IPSO FACTO 22 The best artical was A FLOATING POINT MATH. PACKAGE. by Wayne Bowdish, George Tomczak, and Ron Verlaan.

We are going to run advertisements in IPSO FACTO commencing with the first fall issue. All issues for the next year will be mailed in the first week of every other month. Commencing with October.

### ADVERTISING POLICY

Members may still advertise their personal equipment free.

Advertisements for multiple items of Software, Hardware, and Components will be clased as commercial and will be charged the commercial rate.

### ADVERTISING RATES

Our rates are based on our circulation and type of publication.

Tentative rates are as follows.

1 Full page	\$100.00
1/2 page	\$50.00
1/4 page	\$25.00

Minimum of 1/4 page.

All copy must be camera ready and be accompanied by a certified cheque or money order. Ennyone requiring more information may contact our Advertising Manager

FRED PLUTHERO  
c/o Assec'n of Computer Experimenters  
c/o Bernie Murphy  
102 McCraney street  
Oakville, Ontario  
Canada L6H 1H6

### EXECUTIVE FILE

For our FORTH enthusiasts we are working on FORTH for the ACE system. When it is done and working we will supply it to those who are interested for a printing and mailing fee.

### CLUB HARDWARE PRODUCTS

The club has had several requests recently to provide more information on the ACE hardware projects.

The club has produced, and maintains in stock, 44 pin KLUGE (wire wrap) boards, an 8k EPROM (2708) board, a VIDEO DISPLAY UNIT (6847) board and a buffered 12 slot BACKPLANE conforming to the club's buss. The boards are 6x9.5 inches, plated through, and reflowed. The boards are produced by a commercial circuit board manufacturer. Please refer to the last page of the newsletter for prices and ordering information, and to the last pages of IPSO FACTO issue 21 for the club standard buss.

## NEW PRODUCT ANNOUNCEMENT ---- NETRONICS ELF II TO ACE BACKPLANE ADAPTER<sup>1</sup>

Now netronics ELFII owners can use ACE boards with the new Netronics to ACE adapter.

The NAB is designed to plug into the underside of the ELF II motherboard and provide:

- 6 ACE 44 pin buss slots
- 2 modified 86 pin slots
- 4DB 25 connector circuits
- Serial I/O with optional TTL or RS 232c drivers.
- System power buss distribution.
- price \$ 1.00

The 7.5 x12 inch board is designed to plug into a 86 pin edge connector soldered to the bottom of the existing edgeconnector at buss slot 4 or 5 on the ELFII.

The ELF II is elevated to a 45 degree angle(makes reading the HEX LEDS and using the HEX PAD easier) and the NAB plugs in behind, supporting the mother board and its self. Club boards plug into the NAB and lay parallel to the motherboard. The original blue metal case for the ELF II is retained, but the base is discarded. The new ELF II occupies a 12 inch square area.

The two 86 pin edge connectors are ideal for personal applications, such as serial interfaces, uarts etc(I use mine for a hardware interface to a Quest Super Expansion board). All Netronics signals are present, and one slot provides the signals on both sides of the edgeconnector circuit to facilitate making homemade boards.

The serial/DB 25 connector circuit is flexible enough to accommodate just about any combination of periferall I/O devices you may use. Both inverted and normal signal levels may be used, in combination if needed.

Boards are in stock !

### PROJECTS IN THE WORKS

Currently, ACE is working on three new boards for the club buss. Tony Hill is redesigning the 2708 Eprom board to accommodate 2716 Eproms. The board will provide 16k, in two 8k blocks, and will also have an EPROM BURNER to help you keep your favourite programs. THE board will be available in the fall.

Nearly completed is the 8" Disk Controller Board Project. Actually, the board works fine and is in stock, but because the DMA circuit of the ELF II and QUEST ELF is dedicated to the HEXPAD, it will only work on the TEC 1802 at the present time.

Don't give up, the finest minds? in the club are working on an adapter for the other micros. Look for an appropriate announcement in the next issue of IPSO FACTO.

Finally, Don MacKenzie is ringing the final bugs out of a 32 k dynamic board. This board will use 4116 dynamic rams, and with current prices, club users will be able to add 32k of RAM to their systems for about \$125.00. Look for an announcement in the next issue of IPSO FACTO.

The club executive is considering projects for the next club year. One idea being considered is a new advanced 1802 board. Most of us use one of the commercial 'trainers', hexpad, leds etc. for I/O, and limited interface capability.

We would be interested in your comments and ideas. Please write to Bernie and let us know if you would support a new board, what you want on it, what it should support.

Tied in to the hardware aspect of our commitment to the 1802, we are interested in developing a club standard monitor. Most of the comments we have received on this topic suggest we adopt Steve Nies version 2 of THE MONITOR. This proposal is currently being evaluated by Wayne Bowdish. Please write to us and let us know your ideas, and your support for this project. The club will consider selling the club monitor on an appropriate medium if there is sufficient interest.

In order to facilitate the clubs expansion efforts in hardware and software next year, a new position is being created in the executive - project coordinator. With the clubs year coming to an end, the new executive will be elected in the next few weeks. Again, write with your suggestions for new areas of activity.

JOHN WARE'S SOLUTION TO THE ELF II'S Short Memory.

M.E. Franklin, Milton Ont., June 1981.

I recently purchased a 16k 2114 static memory board from John Ware, 2257 6th Ave., Fort Worth Texas, 76110, phone 817-924-9506.

I am very pleased with the board, and recommend it to an ELF II owner interested in adding static memory to his micro. John sells the bare board for \$35.00, and provides adequate documentation to assemble it.

The board is well designed and laid out, and quite well made for a "home built" product. The board's circuitry employs CMOS throughout, and has provision for on board regulation if needed. It is the same size and pin out as Netronic's boards, and like Netronics products, provides no buss buffers. Like most 16k boards, it uses a 4 to 16 decoder to address one of four 16k memory blocks.

The board is a good and economical addition to my ELF II, and I recommend it to other club members. I suggest you call John first to make sure he has them in stock, and to confirm the current price.

HELP

Claudio Pugliese    Lituania 5457 (1431)    Buenos Aires    Argentina

Claudio would like copies of the following articles from Dr Dobbs Journal

- (1) Use a prom for a Character Generator  
Vol. 2 No.5 p. 17 May 1977 by David Allen
- (2) A Practical Low Cost Home/School UP System  
Vol.2 No.5 pgs.34-44 May 1977 by J. Weisbecker
- (3) Utilities and Music on The Cosmac Elf  
No.19 Vol.2 Issue9 p.30-33
- (4) Programable 1K RAM plus 256 EPROM plus cassette  
Recorder Vol.2 No.19 3&4 by Ed McCormick

Letters of Contact

Robert Passafiume, 3650 1/2 Marlborough, San Diego,  
Cal. 92105.

Would like to contact members on the west coast  
primarily So. Calif.

ERRATA - 'The Monitor - Version 2' 1P50 NO 20

There are four typographical errors in the listing.  
Change location 1 9B from a 72 to a 73.  
Change locations 0 A2 and 0 A3 from 3A 95 to 32 8D  
Change locations 0 A6 and 0 A7 from 3A 95 to 32 8D  
Change location 0 A9 from a 8D to a 95

## 6 FOR SALE

QUEST SUPER ELF with super board, Neutronics key board, Model 40 Teletype, Power supply (Quest). Asking \$200.00  
Dennis Battocchio 1305 Ontario St. Apt.602 Burlington Ont. Canada  
Phone after 5:30P.M. (416)637-5573 L7S 1Y1

**FOR SALE: Quest SUPER ELF 1802 system in cabinet. 4K**

**expansion board, monitor & tiny BASIC ROM, power supply, rf**

**mod, ASCII keyboard, many tapes ,manuals and magazines**

**including Ipso Facto 1-18. All for only \$300.**

**Richard Moffie 20121 Leadwell St. #3 Canoga Park CA. 91306 U.S.A.  
phone (213) 341-6098**

**FOR SALE: ELF II rev.C with Giant Bd. interface, 3ea. 4K memory Bds. (12K total RAM ), 5amp power supply, FULL BASIC on cassette with RPN Math Bd., ASCII keyboard and Video Display Bd., All in Netronics cabinets. Full Documentation and Manuals for machine language and FULL BASIC. Software on cassettes and many 1802 based newsletters.**

**Complete ELF II Computer System- asking \$650 or best offer.**

**Kevin Mast 308 Jackson Ave. Defiance Ohio 43512 usa  
phone (419)782-6147**

**3K Static RAM, compatible with TEC1802. Coded for 0000-0BFF. Ceramic 2114's and all ic's on sockets.**

**Asking \$90.00. Colin Nicholson, 19 Windermere Crt.,  
Brampton, Ontario, Canada. 16X 2L5**

**2708 EPROM board for the "ACE" bus as advertised in  
IPSO FACTO. Unused PCB with 4 24-pin sockets installed  
and includes 3 unused 2708 Eproms. \$25 U.S.  
Tom Jones, 409 Springdale ave., Enterprise, Alabama, 36830**

### **FOR SALE**

**1 Netronics 4K memory board  
without 2102 RAM's. Includes  
DIP switch addressing, fully  
socketed. Asking \$25 (US) or  
best offer. Send SASE to:  
David Schuler, 3032 Avon Road,  
Bethlehem, Pa. 18017, USA.**

## SELL OR SWAP

**TEKTRON 1802 SYSTEM consisting of Tektron 1802 board, Tektron MB1 3/4 K memory with 1 page of CMOS, MB2 7K memory board, 33KSR Teletype, Teletype UART interface board, Keyboard, Keyboard and cassette interface board (incomplete), ACE VDU-Memory board with chips and sockets (not assembled), Hammond case wired with 6 22 pin sockets, Heat sinked regulated on case All tested and complete except for Video board and keyboard interface Asking \$400.00 or will consider HI-FI components. Mike Pupeza 644 Bathurst St. Toronto ontario Canada M5S 2R1 phone (416)535-4127**

Steve Nies  
2510 Deas Street  
Bossler City, LA 71111

## The Text Editor

After finishing work on my last major project, 'The Monitor', I realized after hand coding the entire 2K program how nice it would be to have an assembler. However, before I could use an assembler, I had to have some way of editing the source text. After using IBM's full screen editor last summer, I decided to include this feature in my editor. The major advantage of this method is that line numbers are not needed. Instead, the entire screen is filled with a page of text. If you need to correct a word, all you have to do is move the cursor over the word in error and type in the corrections. The source text and the screen are updated at the same time.

A second advantage of this editor is that any terminal can be used. Even though the length of a line is 80 characters, the screen can be formatted to appear as small as 3 lines of 1 character or as large as 24 lines of 80 characters. If the horizontal width of the screen is less than 80 characters (mine is 32, using the S68047), the entire screen will scroll right or left to allow the user to edit the entire line. I'll mention more about this feature later on in the article.

An example of the screen format is shown in figure 1. Notice that the example is of a 32 character display. The first line of the display is used to enter commands to the editor. Following the command line is what I call a scale line. Besides separating the command line from the source area, a function of the scale

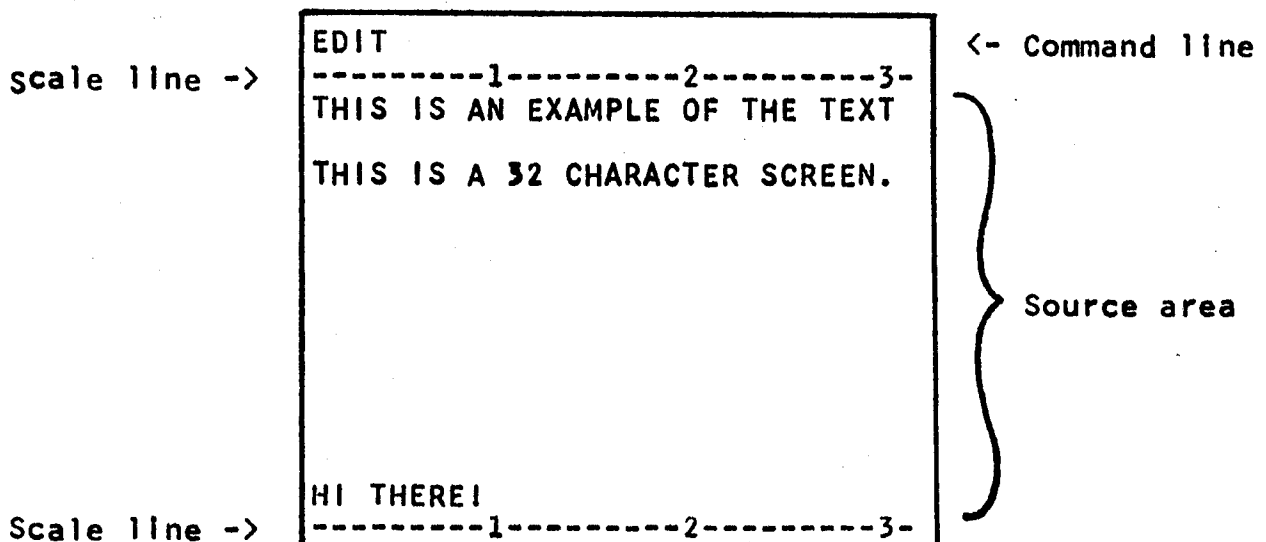


Figure 1

line is to indicate which column the cursor is in. The columns are labeled by tens, so only the tens digit is indicated. The units digit is indicated by counting the number of hyphens since the last number. The area between the scale lines is the source area.

It is in this area that text is entered and corrected. This area will expand and contract depending on the amount of text entered. The last line is another scale line to delimit the display of the source area from the rest of the screen.

There are basically two ways to use the editor. The first is by loading the editor into RAM and then establishing a temporary command to call it. This is done by using the memory examine command to change the end-of-table vector in SYSTEM RAM to point to location Q 00 of the editor. The second way is the method I prefer. All that is required is to place the editor in ROM and then locate the editor ROM directly behind the monitor ROM. This method does not require changing any vectors at all.

There are several options for calling the editor. These options are summarized in Table 1. At this point it might be helpful to discuss some facts about the editor.

COMMAND	DESCRIPTION
TEXT↓ INIT? Y 1000↓	This sequence will initialize the text area as well as the terminal's parameters to the default values. The amount of memory assigned to the text area has been specified at 1000 (HEX) bytes.
TEXT↓ INIT? Y↓	Same as above except that since the end of memory was not entered, the editor will perform a non-destructive search to find the physical end of RAM. All of the available RAM is then assigned to the text area.
TEXT↓ INIT? N 1000↓	This sequence is used to allow the text area and terminal parameters already established by an earlier editing session to still be in effect. Essentially this command is a warm start. Notice that the end-of-memory parameter has been changed to 1000 bytes (HEX).
TEXT↓ INIT? N↓	Same as above except that the editor will assign all of the available memory to the text area.

Table 1

First, the editor assumes that the text starts at location 0000 and works its way up into higher memory. It is possible to set the maximum amount of memory you would like to use as text. This feature can be used to stop text from writing over any programs that are higher up in memory. If you don't enter in the maximum amount of memory to be used, the editor will do a non-destructive search to determine the physical end of memory. All of the available RAM will

COMMAND	DESCRIPTION
TEXT 'file name' 1000 ↵	This command will call the text editor and then load the desired file into the text area. This file must be an ASCII file ONLY! The end of the text area has been specified to be at location 1000 (HEX). The terminal's parameters are also initialized to their default values. This command will perform similar to a cold start.
TEXT 'file name' ↵	Same as above except that all of the available memory has been assigned to the text area.

Table 1 (continued)

then be used as the text area.

Second, if you tell the editor to initialize the text area, it will insert 13 CRs followed by a 00 byte to signal the end of the file. Before a line is edited, it is expanded (excess blanks are inserted on the right) until the record length is 80 characters. After the line is edited, all excess blanks on the right are removed. This allows a line to be stored in RAM with greater efficiency. This process of expanding and contracting a line is invisible to the user. After the text area is initialized, the terminal's parameters are stored in SYSTEM RAM. The default terminal size is 16 lines of 32 characters. These values are recorded at locations Q 4B and Q 4C. If your terminal is a different size, place the screen's horizontal size in HEX minus 1 at location Q 4B. Similarly, place the screen's vertical size in HEX minus 3 at location Q 4C. If you don't use the correct values for your particular terminal, the display will do all sorts of strange things.

Finally, the editor has a feature that could use a bit of explanation. I will be using the editor for some word processing applications, so I need to know when I am coming to the end of a physical piece of paper. With this thought in mind, I developed a feature that would print a line of dots across the screen when you are approaching the end of a physical page. This way, when you are printing a file and the printer hits this line, a form feed will be generated. The printer will then continue printing as normal. (The line of dots will not be printed.) The editor will default to 60 lines of text per page. If you would like to change this value, change the byte at location Q 4A to the desired default value.

The page dividing line will not affect operation of the editor in any way. If you need to move the cursor from one physical page to the next physical page, the cursor will simply hop over the page dividing line. This feature tends to complicate the editor program somewhat, but I feel that the advantages outweigh the extra code required.

The rest of this article will list the commands associated with the editor along with a short explanation on each.

## 1) EDIT

This command is probably the most used command in the entire editor. Its function is to allow the user to enter and correct text. Basically, there are two modes that the editor can be in when using this command. These modes are called the normal and get\_parm modes. The normal mode is when the user wants to enter or correct text. The editor is in this mode most of the time. The get\_parm mode is used when you need to get parameters for some of the commands (ie. From and To addresses for the SAVE command). This mode is indicated by changing the hyphens of the scale line to colons. In this mode, all you can do is move the cursor around. Any command or subcommand that modifies text is deactivated. After the command obtains the necessary parameters, the screen will revert back to the normal mode. I'll describe more about the get\_parm mode later on in the article.

The edit command has several subcommands. These subcommands are all generated by using the keyboard's control characters. In cases where the keyboard has a special key devoted to a control character, I will call that character by name. All other control characters will be denoted by a bar over an ASCII letter. This will indicate that you need to press the control key and the letter to generate the subcommand.

Each of the EDIT subcommands will be listed along with a short explanation on each. But first I would like to mention some notation I will be using in the descriptions.

First, if the control character has arrows around it (for example <S>), this means that this particular subcommand is deactivated while in the get\_parm mode. Second, if a control character is underlined, this means that the bell will be rung if either you use this subcommand at the end of a line or if the record is full. In listing the subcommands, I will first list the name, then the hex representation, and then the key that should be pressed. The explanation will then follow. Now that the notation is discussed, on to the subcommands!

- A) Cursor right,09,HT (or TAB)
- B) Cursor left,08,Backspace
- C) Cursor up,05,U
- D) Cursor down,0A,Line feed

These four subcommands are fairly self-explanatory. However, one thing should be mentioned at this point. For the text editor to work properly, the terminal must be capable of supporting the following cursor movements: CR,LF,BS,TAB (one horizontal space),VT (similar to a reverse LF), and HOME. For those of you that are using the video driver contained in The Monitor, the editor will automatically add the remaining functions (VT and HOME) to the driver.

- E) CRLF,0D,CR

This command will move the cursor to the start of the next line. If the cursor is at the end of the text file, the file will be expanded by 13 blank lines. This expansion will also occur for the cursor down subcommand. If the available text area is full, the bell will ring.

F) Escape edit, 01,  $\overline{Q}$ 

This subcommand is used to exit the edit routine and to move the cursor up to the command line.

G) Set screen, 13,  $\overline{S}$ 

This is used to move the cursor to any column on the screen that is a multiple of ten. After pressing S, the editor will wait for a number between 1 and 8 to be entered. After this number is entered, the cursor will move to the desired column. Entering anything besides 1 through 8 will cause the cursor to be moved to column 1 of the same line.

H) Save parameter, 10,  $\overline{P}$ 

This subcommand is used in the get\_parm mode. It's purpose is to indicate which line should be used as a parameter. For example, after entering SAVE, the command will be expecting both a FROM and TO address. Therefore, the editor will enter the get\_parm mode. The cursor is then moved to the line that is to be used as the starting address. The P key is then pressed to send the address of this line to the save command. The same procedure is used to get the TO address.

Even though this key was meant to be used in the get\_parm mode, it is still active in the normal mode. Pressing this key in the normal mode will take you back to the monitor (without clearing the screen). Even though you can do this, it is not a recommended procedure.

I) Delete line, 0B,  $\langle \overline{K} \rangle$ 

This subcommand will delete the line where the cursor is located from memory.

J) Insert line, 0C,  $\langle \overline{L} \rangle$ 

This is used to insert a line where the cursor is located. The lines below the cursor are moved down to make room for the new line.

K) Delete character, 04,  $\langle \overline{D} \rangle$ 

Pressing this key will remove the character under the cursor from the line. The remainder of the line will move to the left to fill up the resulting hole.

L) Insert character, 03,  $\langle \overline{C} \rangle$ 

This subcommand will move the line one position to the right and insert a blank where the cursor is located.

M) Erase till the end of line, 05,  $\langle \overline{E} \rangle$ 

Used to erase the line starting from where the cursor is located until the end of the line.

## 2) QUIT↵

This command will transfer execution from the editor back to the monitor.

By the way, I forgot to mention that commands may be abbreviated. Look at the Comm\_table in the editor to determine the minimum abbreviation.

## 3) PB↵ or PB digit↵

If a digit was not entered along with the command, the display will move back one page. If a digit was entered, the display will move back that number of pages. If the display is at the start of the file, this command will have no effect.

## 4) PF↵ or PF digit↵

This command is similar to the PB command except that the display moves forward. If the display is at the end of the file, this command will have no effect.

## 5) TOP↵

Moves the display to the start of the file.

## 6) BOTTOM↵

Moves the display to the last page in the file.

## 7) SAVE↵

This command will save a text file on cassette. After entering this command, the screen will go into the get\_parm mode. This indicates that the SAVE routine expects a FROM parameter. If you wish to exit the get\_parm mode at any time, move the cursor to the command line (By using Escape edit) and then type QUIT↵.

In order to get the FROM address, move the cursor to the first line that you wish to save and then press P. Similarly, to obtain the TO address, move the cursor to the last line that you wish to record and press P. The editor will then clear the screen and print "ENTER FILE NAME ->". Enter the desired file name along with a CR. After the save routine has finished recording the file, it will do a fixed delay in order for the operator to read any messages. The display will then return to the normal mode displaying the start of the file.

## 8) LOAD↵

This command will only work in the normal mode. Its purpose is to load a text file from tape and concatenate it to an existing file. If desired, the existing file may simply consist of a single blank line. After entering this command, the editor will prompt you for the file name and then proceed to load the file. After loading is complete, a fixed delay will occur to allow the operator to read any messages. The screen will then display the start of the file in normal mode.

## 9) VERIFY ↵

After using the SAVE command, this command should be used to guarantee that the file was transferred to the cassette properly. If it wasn't, a message will be printed indicating where the error occurred. After a fixed delay occurs to allow the operator to read any messages, the screen will then be cleared and the first page of the text will be displayed.

## 10) PRINT ↵ or PRINT address ↵

This command will print the desired section of text on an output device. After entering this command, the screen will go into the Get\_parm mode to indicate that it expects both FROM and TO addresses. To enter these addresses, simply move the cursor to the desired line and press ctrl P (P). The editor will then print the text starting with the FROM line ending at the TO line.

Another feature of this command is that you can specify the output device desired. Entering PRINT ↵ without an address will select the default value specified at location X 05 (usually set to a printer). To specify a different output device, enter the starting address of it's software driver routine.

At this point I would like to point out one fact concerning the PRINT command. When the text being printed reaches the end of a record, only a Carraige Return is printed. This is because my printer (a Selectric) will perform both a CR and a Line Feed upon occurrence of the CR character. If your printer requires use of the LF character, simply call a little routine that tests for a CR while passing all other characters. If a CR is found, then output both a CR and a LF.

After the text is printed, the output vector that was in effect before this command was used is restored back into SYSTEM RAM.

## 11) FIND /text string/ ↵

The purpose of this command is to locate an occurrence of a string in the text area. If a match is found, the screen is adjusted so that the matched line is the first line on the screen. If a match was not found, the editor will print 'NOT FOUND'.

Another feature of this command is the availability of a wild card character. Using the character '?' in the text string will allow this position to match with any character. For example, if we type FIND /I L??E THE 1802/ and had two lines in the text area that were I LIKE THE 1802 and I LOVE THE 1802, both lines would match. This little feature has come in handy several times!

One fact concerning this command is that the search starts from the second line shown on the screen until the end of the file. If you would like to search the entire file, you must move the screen to the start of the file. This feature will allow a user to selectively search part of the file for a text string.

## 12) MOVE ↵

If it is needed to move a block of text from one location to a new location, the user can use the MOVE command to accomplish this. After MOVE ↵ is entered, the screen will go into the Get\_parm mode to get the FROM, TO, and NEW LOCATION addresses. If the NEW LOCATION address is between the FROM and TO addresses, the bell will be rung. After the block of text is moved to it's new location, the screen will display the first page of the text area.

## 13) COPY ↵

Similar to the MOVE command except that a copy of the block is moved to the new location. The block itself is not moved. This command will not work in the Get\_parm mode.

## 14) CHANGE /text string1/text string2/ ↵

This command will change every occurrence of the first string into the second string. The two strings can be of any length, not necessarily the same length. Only those lines past the second line shown on the screen until the end of the file are checked for a possible occurrence. If the string was not found, the editor will print 'NOT FOUND'. Notice that this command is deactivated in the Get\_parm mode.

This about covers the description of the text editor. For those people who would like to know more about the "Innards" of the editor, I have included a table of useful information at the end of this article. At this point I would like to mention that while the editor has been tested to make sure it will work, all possible combinations of commands and subcommands have not been verified. If any problems arise, I would appreciate it if you would send me a postcard explaining what happened and what you were doing at the time. I will then try to figure out why the problem happened and will take steps to fix it.

-----

I am starting a file system of people who are using any of the software I've submitted to IPSO FACTO. I would appreciate it if anyone who uses this software would send me a postcard indicating what software you are using. Also please briefly describe your system (amount of RAM and ROM, type of terminal, etc.) and indicate any needs for the future. I need this information this information to help me in designing programs for 1802 users. Currently I am in the process of writing an interactive assembler for the 1802. This assembler will have the capability of using a linker/loader to generate relocatable object files. Future plans include a compiler to translate TRS-80 level 2 Basic into 1802 machine language and a Robotic Control Language compiler. I will need information about the systems of individual users to aid me in tailoring the software to fit the users needs. Please address the postcards or letters to the address given at the start of this article.

-----

P.S. One fact that I neglected to mention concerns users with video displays of less than 80 characters per line. As text is entered at the edge of the display, the screen will scroll to the left. However, it is possible for text to be entered faster than the screen will scroll. There are two options available at this point. Either type slower than the screen will scroll (not very practical), or use the Set\_screen subcommand to move the right side of the screen all the way over to the left. Text can then be entered normally.

## EDITOR FACT SHEET

	R(X).1 / R(X).0	Byte	Function
R(0) -	Not Used	10 <u>P</u>	- Save parameter
R(1) -	Temporary	13 <u>S</u>	- Set screen
R(2) -	Stack	01 Q	- Escape EDIT
R(3) -	PC	08 BS	- Cursor <--
R(4) -	Call	0A <u>LF</u>	- Cursor ↓
R(5) -	Return	05 U	- Cursor ↑
R(6) -	Return Addr.	0D CR	- CRLF
R(7) -	Temporary	<u>09</u> HT	- Cursor -->
R(8) -	Screen Home ptr.	0B ( <u>K</u> )	- Delete line
R(9) -	Input Buffer	0C ( <u>L</u> )	- Insert line
R(A) -	Memory Access Subr.	<u>04</u> ( <u>D</u> )	- Delete character
R(B) -	Temporary	<u>03</u> ( <u>C</u> )	- Insert character
R(C) -	Temporary	<u>05</u> ( <u>E</u> )	- Erase till end of line
R(D) -	Text position ptr.	<u>20</u> → <u>7F</u> ( )	- Text characters
R(E) -	Line Ctr. / Screen Pos. Ctr.		
R(F) -	Saved accum. / Scroll Ctr.		

Note: Cntl chars. with parenthesis around them indicate deactivated in Get\_parm.mode.  
 Cntl chars. that are underlined will ring the bell if at the end of the line

## SYSTEM RAM USAGE

S 90

End memory	Page size	Screen hor. size	Screen vert. size	Extended Output	Screen type
------------	-----------	------------------	-------------------	-----------------	-------------

20 -- 22 - 24 -- 26 --- 28 ----- 32 ----- store  
 21 -- 23-- 25 -- 27 --- 29 ----- 33 ----- fetch

## MEMORY ACCESS ADDRESSES

## /\* COMM\_TABLE EXTENSION \*/

Q 00 54 45 00 Q 08  
05 01 Z 00

## /\* MAIN \*/

```

08 D4 I E8 S 6B Y DB 68      Init Output extension vector
10 D4 N 78 3B 20                Call Mon.Quote_search
15 B1 BD AD                      If found quote, load the desired file
18 D4 N 99                      Call Mon.Load
1B F8 00 5D 30 45                Insert an end-of-file mark in the text
20 D4 I DD 49 4E 49 54 3F A0    Print "INIT? "
29 F8 00 A9                      Init the buffer pointer
2C D4 J 05                      Call Mon.Inbuff
2F F8 00 A9 09 FB 59 3A 4E      Test to see if the first char. is a 'Y'
37 BD F8 0D AD 52 F8 00 5D      Yes, so init the text area with CRs
3F 2D 02 5D 8D 3A 3F
45 D4 I E8 S 92                Call Mon.Ram-init
4A 3B 1F 0D 68                Init video terminal parameters
4E D4 J A0 33 65                Try to get a end-of-memory parameter
53 F8 FF BC AC                Parameter was not specified, so find
57 C8 02 5C                    the end of memory with a non-
5A 1C 0C 52                    destructive test
5D F8 AA 5C 0C FB AA 32 58
65 93 BA F8 93 AA                Init Mem Access subroutine PC
6A 9C DA 20 8C DA 22            Store end-of-memory address at loc. S 90
70 F8 00 DA 32                Init Type_of_screen flag
74 B8 A8 BD AD BE AE AF        Init text editor parameters
7B D4 U 19                      Call Fix_FF
7E D4 Q EE                      Call Display_screen
81 D4 I DD 84                    Print a HOM character
85 D4 Q A0                      Call Buff_init
88 D4 J 05                      Call Mon.Inbuff
8B D4 J 2E Q AB                Call Mon.Comm_rec
90 30 81

```

## /\* MEM ACCESS \*/

```

92 D3
93 52 43 F6 F9 80 A9            Save accumulator and get index
99 02 CF 59 38 09 30 92        Either save or load D reg., depending DF

```

## /\* BUFF\_INIT \*/

```

A0 F8 40 A9                    Init ptr to end of buffer
A3 29 F8 20 59 89 3A A3        Insert blanks into buffer
AA D5

```

```

/* COMM_TABLE */
Q AB 45 00 R A9      (Edit) Protected
AF 51 00 U D9        (Quit)
B3 50 42 00 W 48      (Ph)
B8 50 46 00 W 5B      (Pf)
BD 54 00 W 92         (Top)
C1 42 00 W 99         (Bottom)
C5 53 00 W A3         (Save)
C9 4C 00 W B2         (Load) Protected
CD 56 00 W C9         (Verify)
D1 50 52 00 W FD      (Print)
D6 46 00 X D2         (Find)
DA 43 4F 00 X EA      (Copy) Protected
DF 4D 00 X F0         (Move)
E3 43 48 00 Y 87      (Change) Protected

```

```

/* CRLF */
E8 D4 L DD OD 8A      Print a CRLF
ED D5

```

```

/* DISPLAY_SCREEN */
EE D4 L DD OC 8A      Print a FF and a LF
F3 D4 R 5F            Call Scale_line
F6 D4 Q E8            Call CRLF
F9 98 B7 88 A7        Put the screen home address in R(7)
FD DA 29 AC           Get the terminal vertical size
R 00 8F BC            Save the Scroll ctr reg. in R(C)
02 DA 27 AB           Get the terminal's horizontal size
05 47 32 33          Get a character to be displayed
08 FB OD 32 2A        Test for the end of a record (CR)
0C FB 01 32 17        Test for the end of a record (LF)
10 FB 0C D4 R 52 30 05 Char. was not a control char., print it
17 D4 Q E8            Call CRLF
1A 2C 8C 32 2E        Finished printing screen?
1E DA 27 AB           Get the terminal's horizontal size
21 F8 2E D4 S 67      Print the screen dividing line
26 2B 8B 3A 21        Continue until one whole line is printed
2A D4 Q E8            Call CRLF
2D 2C                Decrement vertical size counter
2E 9C AF              Restore Scroll counter
30 8C 3A 00           Finished printing screen?
33 D4 R 5F            Yes, so call Scale_line
36 D4 L DD 04 8A      Print a Home and a LF
3B 9E AC 38 2C        Move the cursor down until the temp.
3F D4 L DD 8A         line ctr equals zero
43 8C 3A 3E
46 8E AC
48 8C 32 55
4B D4 L DD 89
4F 2C 30 48

```

```

Move the cursor to the right until the
temp screen position ctr equals
zero

```

```

/* SCREEN_PRINT */
R 52 8F CE 2F D5      Skip print if temp.scroll not = 0
56 8B 32 55          Skip print if past screen end
59 2B 9F D4 S 67 D5    Otherwise, print the character

/* SCALE_LINE */
5F F8 30 BB          Init column indicator to ASCII zero
62 8F BC              Save contents of scroll counter
64 DA 27 AB          Get terminal's horizontal width
67 F8 0A AC          Init column's digit counter
6A 2C 8C 32 7F       See if a digit or a hyphen is printed
6E DA 33 32 75       Select either a hyphen or colon
72 F8 3A C8 F8 2D
77 D4 R 52 3A 6A      Call Screen_print
7C 9C AF D5          Restore scroll ctr and return
7F 9B FC 01 BB       Add one to the column tens digit
83 D4 R 52 3A 67      Call Screen_print
88 30 7C

/* TEST_EOF */
8A 2D                Decrement text pointer
8B 0D 32 94          Test for an end-of-file byte
8E FB 0D 32 94       or a CR
92 FB 01 D5          or a FF

/* START_OF_RECORD */
95 8D 3A 9B 9D 32 A1 Exit if at location 0000
9B D4 R 8A 3A 95      Call Test_eof
A0 1D D5

/* END_OF_RECORD */
A2 1D                Increment text pointer
A3 D4 R 8B 3A A2      Call Test_eof
A8 D5

/* EDIT */
A9 98 BD 88 AD        Set the text ptr to screen's home addr
AD D4 I 96 38 1F 38 2F Call Extend
B4 D4 Q EE 30 BD      Call Display_screen
B9 1D
BA D4 I A8            Call Mon.Bell
BD D4 S 64 3B BD      Get a character from the keyboard

/* ESC_EDIT */
C2 FB 11 3A CD        Test for a Q
C6 BE AE AF          Init Screen position ctr, scroll ctr,
C9 D4 U 41            and Line ctr. Then call Compact
CC D5

/* CURSOR ← */
CD FB 19 3A E6        Test for a Backspace character
D1 8D 3A D7 9D 32 BA  Ring bell if at location 0000
D7 D4 R 8A 32 B9      Call Test_eof
DC 8E 32 B3          If Screen pos. ptr. = 0, dec. scroll
DF 2E                Decrement Screen position counter
E0 D4 I DD 88 30 BD    Move the cursor to the left

```

```

/* CURSOR → */
R E6 FB 01 3A FE      Test for a Tab character
EA D4 R 8B 32 BA      Call Test_eof
EF 1D                 Increment text position ptr
F0 DA 27 52           Get the terminal's horizontal width
F3 8E F7 33 B1        Inc. Scroll ctr. if past the edge
F7 1E D4 I DD 89 30 BD Increment cursor position

/* CURSOR ↓ */
FE FB 03 3A 08        Test for a Line feed character
I 02 D4 U 66 C0 R BD   Call Cursor_down

/* CURSOR ↑ */
08 FB 1F 3A FE        Test for a U
0C 8D 73 9D 73        Save the text position pointer
10 D4 R 95             Call Start_of_record
13 8D 3A 19 9D 32 45   Ring bell if at location 0000
19 2D D4 I 96 27       Call Extend
1E 12 12 87 FD 50 F5 73 Determine address of next line up
25 02 7F 00 73
29 D4 R A3             Call End_of_record
2C 9E 32 4A           Test to see if we are at top of page
2F FF 01 BE           No, so decrement line counter
32 D4 I DD 8B         Print a Vertical Tab
36 4D FB 0C 32 2C     Do we need to skip over a dotted line?
3B D4 U 41            Call Compact
3E 12 42 BD 02 AD 30 05 No, so restore text position ptr
45 D4 I A8 30 3E      Call Mon.Bell
4A 1D AC 1C           We need to move the display back a
4D DA 29 FF 01 BE     page, so move the cursor to the
52 D4 I 57 30 3B      last line of the preceeding page

/* SCREEN_UP */
57 8D 73 9D 73        Save the text position pointer
5B 98 BD 88 AD        Get the screen's home address
5F DA 29 A7           Get the terminal's vertical size
62 8D 3A 68 9D 32 8B
68 2D D4 R 95         Call Start_of_record
6C 2D 4D FB 0C 3A 76   Do we need to skip over a dotted line?
72 27 87 32 7A       Yes, so decrement the temp. line ctr.
76 27 87 3A 62       Decrement the line counter
7A 2C 8C 3A 5F       Finished moving up pages?
7E 9D B8 8D A8       Update the screen home address
82 D4 Q EE           Call Display_screen
85 12 42 BD 02 AD D5   Restore the text position pointer
8B DA 29 52 87 F5     Calculate line ctr value for less
90 CE FF 01 BE 30 7E   than one page moves

/* EXTEND */
96 8D 73 9D 73        Save the text position pointer
9A D4 R 95             Call Start_of_record
9D F8 00 A7 C8 1D 17   Count the number of characters for
A3 D4 R 8B 3A A1       this line
A8 87 FD 50 A7         Calculate the number of blanks needed
AC F8 20 D4 I B3      Call Move_line
B1 30 85

```

```

I B3 8D 73 AC 9D 73 BC /* MOVE_LINE */
B9 4C 3A B9 2C Save the text position pointer
BD 9C BB 8C AB Find the end of the file
C1 87 52 8C F4 AC 9C 7C 00 BC Add amount of extension to end of file
CA F8 91 A9 E9 Set up a ptr to the end of memory limit
CE 8C F7 29 9C 77 3B DA See if exceeded memory limit
D5 D4 L A8 30 85 Yes, so ring the bell
DA E2 Restore X ptr
DB 0D 52 F8 00 5D 0B Insert a marker to indicate end of move
E1 2B 5C 2C 0B 3A E1 Extend file
E7 02 C8 2C 9F 5C Restore byte where marker was inserted
EC 8C 52 8D F7 9C 52 9D 77 Fill the resulting hole with the filler
F4 3B E9 character contained in R(F).1
F6 9F FD 1F 3B 85 Was the filler char. a CR?
FB C0 U 1D Yes, so jump to Fix_FF

/* CRLF */
FE FB 18 3A E3 Test for a CR
U 02 AE Yes, so zero screen position counter
03 D4 R 95 Call Start_of_record
06 D4 U 66 Call Cursor_down
09 8F 3A 13 Is the scroll counter equal to zero?
0C D4 L DD 8D C0 R BD Yes, so print a CR
13 F8 00 AF C0 R B4 No, so zero it and jump to Display_screen

/* FIX_FF */
19 8D 73 9D 73 Save the text position pointer
1D F8 FF BD AD Set up a ptr to the start of file - 1
21 DA 25 52 Get the lines per physical page limit
24 F8 01 A7 Init a ctr that counts the no. of CRs
27 1D
28 22 D4 R A3 12 Call End_of_record
2D 0D FC 00 32 63 Get the end-of-record character
32 87 F3 32 3C If the number of CRs is below the
36 17 F8 0D 5D 30 27 limit, store a CR
3C F8 0C 5D 30 24 Otherwise, store a Form Feed

/* COMPACT */
41 8D 73 9D 73 Save the text position pointer
45 D4 R A3 Call End_of_record
48 9D BB 8D AB
4C F8 00 A7 38 17 Initialize a temporary counter
51 8D 3A 57 9D 32 5E Count the number of excess blanks
57 2D 0D FB 20 32 50 1D that we can remove from the line
5E 4B 5D 1D 3A 5E Compress the line
63 C0 T 85 Go restore the text position pointer

```

## /\* CURSOR\_DOWN \*/

<u>U</u> 66	8D 73 9D 73	Save the text position pointer
6A	D4 <u>R</u> A3	Call End_of_record
6D	1D 0D 3A 8A	Are we at the end of the file?
71	DA 29 A7	Yes, so get the terminal's vertical size
74	F8 0D 5D 1D F8 00 5D 2D	Set up a new record
7C	0D D4 <u>I</u> B3 3B 88	Call Move_line to store another page
82	F8 00 5D C0 <u>I</u> D5	We exceeded memory, so restore file
88	F8 00 B7	Set a flag to show that we added to file
8B	D4 <u>I</u> 96 33 85	Call Extend
90	12 12 02 AD FC 51 73	Calculate address of the last current line
97	02 BD 7C 00 73	
9C	D4 <u>U</u> 41	Call Compact
9F	12 12 87 F5 73 AD	Get address of the new current line
A5	02 7F 00 73 BD	
AA	D4 <u>R</u> 95 2D	Call Start_of_record
AE	DA 29 52	Get the terminal's vertical size
B1	4D FB 0C 3A C3	Do we need to hop over a dotted line?
B6	9E FC 01 BE F7 33 D4	Yes, so increment line counter
BD	22 D4 <u>I</u> DD 8A 12	Print a LF
C3	9E FC 01 BE F7 33 D4	Increment the line counter
CA	D4 <u>I</u> DD 8A	Print a LF
CE	97 C2 <u>I</u> 82 30 63	If we extended the file, go display
D4	F8 00 C0 <u>I</u> 93	the screen. Otherwise, go display a new page.

## /\* QUIT \*/

D9	D4 <u>I</u> DD 8C	Clear the screen
DD	12 42 A6 02 B6 D5	Modify the return address to jump to the monitor

## /\* SAVE\_PARM \*/

E3	FB 1D 3A F1	Test for a $\bar{P}$
E7	D4 <u>R</u> 95	Call Start_of_record
EA	D4 <u>R</u> C6	Call Esc_edit
ED	FC 00 30 DD	

## /\* SET\_SCREEN \*/

F1	FB 03 CA <u>V</u> 27	Test for a $\bar{S}$
F6	AE AF	Zero screen position ctr and scroll ctr
F8	D4 <u>R</u> 95	Call Start_of_record
FB	D4 <u>S</u> 64 3B FB	Get the column number from keyboard
<u>V</u> 00	FF 31 3B 24	Is the number between 0 and 9?
04	FF 08 33 24	
08	FC 09 A7 F8 09 C8	Yes, so store in the ten's counter
0E	F8 0A AC	Init the one's counter
11	DA 27 52 8E F7 33 1A	If past screen's edge, inc. scroll ctr
18	1E 38 1F	Otherwise, inc. screen position ctr
1B	1D	Increment text position pointer
1C	2C 8C 3A 11	Finished doing unit's movement?
20	27 87 3A 0E	Finished doing ten's movement?
24	C0 <u>R</u> B4	Yes, so display screen
27	DA 33 CA <u>R</u> BA	The rest of the EDIT subcommands are inactive if in the get_parm mode

```

/* DEL_LINE */
V 2C 9F FB 0B 3A 59      Test for a K character
31 8D 73 9D 73          Save the text position pointer
35 D4 R A3 1D           Call End_of_record
39 0D 32 54             Ring Bell if at the end of the file
3C 9D B7 8D A7
40 2D D4 R 95           Call Start_of_record
44 47 5D 1D 3A 44       Delete the record
49 D4 U 19              Call Fix_FF
4C 12 42 BD 02 AD       Restore the text position pointer
51 C0 R AD              Go extend the next line
54 D4 L A8 30 4C        Call Mon_Bell

/* INSERT_LINE */
59 FB 07 3A 74          Test for a L character
5D 8D 73 9D 73          save the text position pointer
61 D4 U 41              Call Compact
64 D4 R 95              Call Start_of_record
67 F8 01 A7
6A F8 0D D4 I B3        Call Move_line to insert a new record
6F D4 I 96 30 49        Call Extend

74 9F BC D4 R 8B C2 R BA  The following EDIT subcommands won't
                          work if at the end of the record

/* TEXT */
7C 9C FF 20 3B 8D       Is the character a letter?
81 9C 5D                Yes, so store it in the text area
83 D4 S 67 D4 L DD 88   Print it
8A C0 R EA              Jump to cursor_right

/* ERASE_LINE */
8D FB E5 3A A0          Test for a E character
91 8D 73 9D 73          Save the text position pointer
95 F8 20 5D 1D          Store blanks in the rest of the line
99 D4 R 8B 3A 95        Call Test_eof
9E 30 BB

/* DEL_CHAR */
A0 FB 01 3A AB          Test for a D character
A4 F8 01 D4 W 26        Call Delete
A9 30 B7

/* INSERT_CHAR */
AB FB 07 CA R BD        Test for a C character
B0 F8 01 D4 V EE 3A 29  Call Insert
B7 8D 73 9D 73          Save the text position pointer
BB D4 R 95              Call Start_of_record
BE D4 L DD 8D           Print a CR
C2 DA 27 AB 8F A7       Get the screen's horizontal size
C7 D4 R 8B 32 D8        Call Test_eof
CC 4D D4 R 52 30 C7     Call Screen_print
D2 2B F8 20 D4 S 67     Blank out the rest of the line
D8 8B 3A D2
DB 87 AF 8E A7          Restore the scroll counter
DF D4 L DD 8D           Print a CR
E3 87 C2 I 3E          If done, go restore text position ptr
E7 27 D4 L DD 89 30 E3 Move cursor to original position

```

```

/* INSERT */
V EE 9F A7 AB      Save the number of char. to insert
F1 0D 73 F8 00 5D 1D Store a marker to indicate finished
F7 D4 R A3         Call End_of_record
FA 9D BC 8D AC 2C
FF 2D 0D FB 20 3A 1F
W 05 27 87 CA V FF  See if we have room to insert chars.
0A 2D 0D 32 12     Make room to insert characters
0E 5C 2C 30 0A
12 12 02 5C
15 2C F8 20 5C 2B 8B 3A 15 Restore byte where marker was placed
1D D5             Fill resulting hole with blanks
1E 2D 0D 3A 1E
22 12 02 5D D5    Couldn't insert, so restore text
                  pos. ptr. and byte where marker
                  was placed

/* DELETE */
26 9D BB BC 8D AB AC
2C 9F A7 A1        Save number of bytes to be deleted
2F 1B 27 87 3A 2F  Set up a temporary pointer
34 0B D4 R 8C 32 3F Call Test_eof
3A 4B 5C 1C 30 34  Delete characters
3F F8 20 5C 1C 21 81 3A 3F Fill the resulting hole with blanks
47 D5

/* PB */
48 F8 01 B7 AC 2C  Init the number of pages to move back
4D D4 J AE         Do we want to move back more than one?
50 8C 3A 54 1C    If 0 pages entered, set to one page
54 D4 I 57        Call screen_up
57 F8 00 BE D5    Zero the line counter

/* PF */
5B F8 01 B7 AC 2C  Init the number of pages to move up
60 D4 J AE         Do we want to move up more than one?
63 8C 3A 67 1C    If 0 pages entered, set to one
67 8D 73 9D 73    Save the text position pointer
6B 98 BD 88 AD     Get the screen home address
6F DA 29 A7        Get the screen's vertical size
72 D4 R A3         Call End_of_record
75 4D FB 0C 3A 7E  Do we need to skip over a dotted line?
7A 27 87 32 85    Yes, so decrement temp. line counter
7E 0D 32 8C        Have we reached the end of the file?
81 27 87 3A 72    No, so decrement the temp. line ctr
85 2C 8C 3A 6F     Finished moving all pages?
89 C0 I 7E        Yes, so go restore text position pointer
8C 2D D4 R 95 30 89 Call Start_of_record

/* TOP */
92 F8 74 A6 F8 00 D5 Jump back to register Init in MAIN

```

```

/* BOTTOM */
W 98 18 08 3A 98      Move the display to the end of the file
9C F8 01 AC          Then move the display back a page
9F D4 I 57           Call screen_up
A2 D5

/* SAVE */
A3 F8 05 D4 X 45 33 DC  Call Get_parm
AA D4 W E0           Call Get_name
AD D4 Q E9 30 CF      Call Mon.Save

/* LOAD */
B2 DA 33 CA I A8      Ring bell if in Get_parm mode
B7 BC AC B1          Start loading at the end of the
BA 4C 3A BA 2C        current file
BE D4 W E0           Call Get_name
C1 D4 N 95           Call Mon.Load
C4 F8 00 5D 30 CF      Write an end-of-file mark in text area

/* VERIFY */
C9 D4 W E0           Call Get_name
CC D4 N 89           Call Mon.Verify
CF F8 DF B8          Do a fixed delay
D2 28 98 3A D2
D6 F8 Q BA F8 93 AA   Restore R(A) to point to Mem_access
DC F8 70 A6 D5        Jump to register init in MAIN

/* GET_NAME */
E0 D4 I DD           Print 'ENTER FILE NAME ->'
E3 0C 45 4E 54 45 52 20
EA 46 49 4C 45 20
EF 4E 41 4D 45 20 2D BE
F6 D4 Q A0           Call Buff_init
F9 D4 J 05           Call Mon.Inbuff
FC D5

/* PRINT */
FD F8 04 B7 D4 J AE 33 0B  Call Mon.Expr4
X 05 F8 P BC F8 66 AC      Init default to screen
0B 9C 73 8C 73          Save output vector
0F F8 05 D4 X 45        Call Get_parm
14 12 42 A7 02 B7 33 42  Restore output vector
1B F8 68 A9            Set up a ptr to the output vector
1E 49 73 09 73        Save the current output vector
22 87 59 29 97 59      Store the new output vector
27 0C A7 F8 00 5C      Insert a marker to signal when done
2C 4D 32 34           Print the file until we hit the marker
2F D4 S 67 30 2C
34 87 2D 5D           Restore the byte where marker was put
37 D4 I DD 8D         Print a CR
3B 19 12 42 59 29 02 59 Restore the old output vector
42 C0 W D6

```

```

/* GET_PARM */
X 45 DA 32      Set screen to get_parm mode
47 D4 Q 7E 33 95 Call MAIN
4C 9D 73 8D 73  Save the first parameter
50 DA 33 F6 3B 73 Finished getting all parameters?
55 59 D4 Q 81 33 93 No, so call MAIN
5B D4 R A3      Call End_of_record
5E 9D 73 8D 73  Save the second parameter
62 DA 33 F6 3B 73 Finished getting all parameters?
67 59 D4 Q 81 33 91 No, so call MAIN
6D 9D 73 8D 73  save the third parameter
71 DA 33
73 59 12 42 AC 02 BC Put the first parameter in R(C)
79 09 F6 3B 8C   Any more parameters?
7D 59 12 42 AD 02 BD Yes, so put it in R(D)
83 09 F6 3B 8C   Any more parameters?
87 12 42 AE 02 BE Yes, so put it in R(E)
8C F8 00 DA 32 D5 Set the screen to the normal mode
91 12 12 12 12 D5 Restore the stack if premature exit

/* FIND_STRING */
96 49 FB 2F 32 A3      Search for the first '/'
9B 89 FF 40 C3 M 46 30 96
A3 E9
A4 89 B7
A6 F8 00 A7
A9 97 A9
AB 4D 32 CC
AE F3 3A AB 19 17
B3 F8 2F F3 32 C4
B8 F8 3F F3 32 C1
BD 0D F3 3A A6
C1 1D 30 B1
C4 87 AB
C6 2D 2B 8B 3A C6
CB D5
CC F6 8B CA N 69 D5

Set X to point to the command line
Save the address of the first '/'
Init the first parm's length ctr
Put the address of the first '/' in R(9)
Are we at the end of the file?
No, so does first character match?
Yes, so test for end of the first parm.
No, so test for a wild card character
No, so see if the text char. matches
Yes, so see if the rest matches
Match found, so back up the text
pos. ptr. to start of match

If nothing was found, print 'NOT FOUND'

/* FIND */
D2 AB 98 BD 88 AD
D7 D4 R A3      Call End_of_record
DA D4 X 96 33 D1 Call Find_string
DF D4 R 95      Call Start_of_record
E2 9D B8 8D A8  Found match, so set screen
E6 D4 Q EE      Call Display_screen
E9 D5

/* MOVE, COPY */
EA DA 33 CA I A8 C8  Entry for Copy
F0 F8 FF 73      Entry for Move
F3 F8 0F D4 X 45 12 C3 W D6 Call Get_parm
FC 02 A1         Put selection flag in R(1).0
FE D4 I F7       Call Mon.Test
Y 01 8C 52 8D F7 22 Ring bell if third parm. is less than
06 9C 52 9D 77 12 3B 17 second and greater than first parms.

```

```

Y 0D 8E F5 22 9E 75 12 33 82
15 FF 00
17 81 7E A1
1A 98 BB 88 AB
1E 4B 3A 1E
21 8B 52 8F F4 A7
26 9B 52 9F 74 B7
2B 2B
2C DA 21 73 DA 23 52
32 87 F7 12 97 77 33 82
39 0C 52 F8 00 5C
3E 57 27 2B 0B 3A 3E
44 02 57 27 5B
48 81 F6 A1 3B 62 1F
4E 8F 52 8E F4 AE 22
54 9F 52 9E 74 BE 12
5A 8D F4 AD 22 9D 74 BD 12
62 9D BC 8D AC
66 0E 52 F8 00 5E 30 70
6D 2C 57 27 0C 3A 6D
73 02 57 5E
76 81 32 7F
79 1D 0D 5E 1E 3A 79
7F C0 W 92
82 D4 I A8 30 7F

```

Shift result into selection flag

Find the end of the text  
Add the length of (parm 2 - parm 1)  
to the end of the text address

Ring the Bell if we exceeded the  
memory size limit  
Insert a marker to signal all done  
Extend the text area  
Restore the byte where marker was put  
If parm 3 < parm 1 & parm 2, then  
update both parms. 1 & 2

Insert a marker byte  
Move the block of text to the new loc.  
Restore the byte where marker was put  
If the command was Move, then erase  
the old block of text  
Go print the first page of the file  
Call Mon.Bell

/\* CHANGE \*/

```

87 AB DA 33 CA I A8
8D A9 98 BD 88 AD
92 D4 X 96 33 DA
97 AC 19 89 B7
9B 49 FB 2F 32 A9
A0 89 FF 40 C3 M 46 1C 30 9B
A9 8C B1
AB 87 D4 W 26
AF 91 32 CA
B2 D4 I 96
B5 97 A9
B7 91 D4 V EE 3A D4
BD 09 FB 2F 32 C7
C2 49 5D 1D 30 BD
C7 D4 U 41
CA F8 00 A9
CD D4 X 96 3B 97 30 7F
D4 D4 I A8 C0 X E2 D5

```

Ring Bell if in Get\_parm mode  
Get the screen's home address  
Call Find\_string  
Save the address of the second '/' + 1  
Determine length of the second parm.

Save the length in R(1).1  
Call Delete  
Skip Insert if null string  
Call Extend  
Get address of the second '/' + 1  
Call Insert  
Insert the second parm into record

Call Compact  
Reset the command line ptr  
Call Find\_string  
Call Mon.Bell

/\* OUTCHAR EXTENSION \*/

```

DB FB EB 3A F1
DF 88 FF 20 33 E9
E4 98 FF E1 3B F9
E9 88 FF 20 A8 98 7F 00 B8
F1 FB 0F 3A F9
F5 A8 F8 E0 B8
F9 9F FF 20 C0 P C5
FF 00

```

Test for a VT character  
See if we can move up a line  
Yes, so move up a line  
Test for a HOM character  
Yes, so move cursor to home location  
Jump back to Outchar routine

## MODIFICATIONS TO THE CLUB VDU BOARD

P Muir

The following are several modifications that I have made to the club VDU board to increase it's flexibility. The accompanying sketch is modified from John Myszkowski's article in IF #18 p32 (or Best of Ipso III-179).

- 1) The mode control chip 4508 is enabled if the address is FCOO-FFFF. By using an 8 input gate, this can be reduced to a range of FFF8-FFFF. This frees up almost 1K of RAM. I used an 8 input NAND gate (4068) on address lines A3 to A9 plus the chip enable previously supplying the 4508 (pin 11 of chip 10). I mounted the chip on a small PC board elevated from the main board in the lower right hand corner using two small bolts and rubber washers. There is a spare inverter available on the 4049 hex inverter. Of course, an 8 input AND gate would simplify the wiring.
- 2) I have reversed the connections for the alphanumerics/semi-graphics and the inverse functions on the 6847 since in the semigraphics-4 mode the current configuration restricted its colour range:  
 GIXX XXXX Alphanumerics                      GCCC LLLL Semigraphics-4
- 3) To get the clear picture required for the high resolution graphics, I am currently running the 1802 at 3.58 MHz which eliminates interference from the 1802 clock and makes the addition of filtering capacitors superfluous. By using a switch it is possible to select 3.58 MHz or a second, slower speed.
- 4) To further enhance the picture, I use direct video input with luminance alone for high contrast black and white or with full composite for colour. The latter is not as sharp in spite of adding trimming pots as seen in the diagram. None of these are needed if luminance is used. The composite video is supposedly obtained by placing a diode between pins 13 and 14 of the 1372; however, the crucial factor appears to be putting a positive voltage on pin 14. Reversing this will invert the signal.

- 5) To increase the signal strength, I have added a transistor inverter and amplifier stage on my home-brew mother board. This was initially built before switching to composite video capabilities but a single stage noninverting amp would probably work as well.

I am currently thinking of revamping a board to set up an external character generator for upper and lower case characters. This will require an 8 bit counter for the row preset and horizontal sync as well as further buffer control. If someone has one running please send a note to Ipso.

I am also developing software for a flexible graphics control since I am interested in having plotting and eventually '3D' graphic capabilities. If anyone is working towards this please drop me a line at the following address.

1552 Lovelady Cr,  
Mississauga, Ontario,  
Canada L4W2Z1



## STATUS DISPLAY

P Muir

The following is the entry program to my monitor which displays the micro status in the following manner:

BREAK	HI LO	0	00 53
		1	FF F8
X	2	2	00 FF
P	3	3	FF 12
		4	B8 01
D	00	5	B8 13
DF	01	6	FE 21
		7	FE 00
		8	FE E5
		9	25 BB
		A	CA EF
		B	EO 00
		C	34 BF
		D	EO 21
		E	E1 FA
		F	20 00

I have only included the store routine since we all have different display routines. The Break address is that at which the monitor has inserted a breakpoint ( 00 00 if none present). Note that this program easily fits in the 1K of RAM made available at FC00-FFFF by revamping the mode addressing. This will be described in another article.

The obvious advantage to the program is that it shows all registers and although the program counter is different, the breakpoint address shows it's previous location.

```

FF 00 79 MARK-X,P
01 E2 SEX-2
02 73 STXD-D
03 7E SHLC
04 FA ANI
05 01
06 73 STXD-DF
07 90 GHI-0
08 73 STXD
09 80 GLO-0
0A 73 STXD
0B 91 GHI-1
0C 73 STXD
0D 81 GLO-1
0E 73 STXD
0F 93 GHI-3
FF 10 73 STXD
11 83 GLO-3
12 52 STR-2
13 F8 LDI
14 FF
15 B1 PHI-1
16 F8 LDI
17 F7
18 A1 PLO-1
19 E1 SEX-1
1A 8F GLO-F
1B 73 STXD
1C 9F GHI-F
1D 73 STXD
1E 8E GLO-E
1F 73 STXD
FF 20 9E GHI-E
21 73 STXD
22 8D GLO-D
23 73 STXD
24 9D GHI-D
25 73 STXD
26 8C GLO-C
27 73 STXD
28 9C GHI-C
29 73 STXD
2A 8B GLO-B
2B 73 STXD
2C 9B GHI-B
2D 73 STXD
2E 8A GLO-A
2F 73 STXD
FF 30 9A GHI-A
31 73 STXD
32 89 GLO-9
33 73 STXD
34 99 GHI-9
35 73 STXD
36 88 GLO-8
37 73 STXD

```

```

FF 38 98 GHI-8
39 73 STXD
3A 87 GLO-7
3B 73 STXD
3C 97 GHI-7
3D 73 STXD
3E 86 GLO-6
3F 73 STXD
FF 40 96 GHI-6
41 73 STXD
42 85 GLO-5
43 73 STXD
44 95 GHI-5
45 73 STXD
46 84 GLO-4
47 73 STXD
48 94 GHI-4
49 73 STXD
4A 42 LDA-2 LO-3
4B 73 STXD
4C 42 LDA-2 HI-3
4D 73 STXD
4E 82 GLO-2
4F FC ADI
FF 50 06 INITIAL POS'N-2
51 73 STXD
52 92 GHI-2
53 7C ADCI
54 00
55 73 STXD
56 42 LDA-2 LO-1
57 73 STXD
58 42 LDA-2 HI-1
59 73 STXD
5A 42 LDA-2 LO-0
5B 73 STXD
5C 42 LDA-2 HI-0
5D 73 STXD
5E 42 LDA-2 DF
5F 73 STXD
FF 60 42 LDA-2 D
61 73 STXD
62 02 LDN-2 X,P
63 51 STN-1
64 D3 SEP-3
65 C0 LBR
66 FC MONITOR LOC'N
67 00

```

D3 AT 64 IS FOR RETURN IF  
 USING REG SAVE ALONE  
 ENTER WITH PC = R0

## STORAGE POSITIONS FOR REGSAVE

FF	D5	23	X,P
	D6	00	D
	D7	01	DF
	D8	00	HI-0
	D9	53	LO-0
	DA	FF	HI-1
	DB	F8	LO-1
	DC	00	HI-2
	DD	FF	LO-2
	DE	FF	HI-3
	DF	12	LO-3
FF	EO	B8	HI-4
	E1	01	LO-4
	E2	B8	HI-5
	E3	13	LO-5
	E4	FE	HI-6
	E5	21	LO-6
	E6	FE	HI-7
	E7	00	LO-7
	E8	FE	HI-8
	E9	E5	LO-8
	EA	25	HI-9
	EB	BB	LO-9
	EC	CA	HI-A
	ED	EF	LO-A
	EE	EO	HI-B
	EF	00	LO-B

FF	FO	34	HI-C
	F1	BF	LO-C
	F2	EO	HI-D
	F3	21	LO-D
	F4	E1	HI-E
	F5	FA	LO-E
	F6	20	HI-F
	F7	00	LO-F
	F8		MODE
	F9		MODE
	FA		MODE
	FB		MODE
	FC		MODE
	FD		MODE
	FE		MODE
	FF		MODE

### Netronics Tiny Basic can run on all 1802 computers

Netronic Tiny Basic is a good interpreter for its size and price. I think that non ELF II 1802 users have looked at this Tiny Basic with some jealousy.

Well, no more jealousy now, for with some patches they also can enjoy Netronics Tiny Basic.

However, you must have 4 K bytes of RAM from M 0000 - M 0FFF.

#### Patches

- A) Netronics Tiny Basic includes a software UAR/T that is connected to the inverted EF 4 flag. That means that when the terminal is on and no key is touched the EF 4 pin of the 1802 is "1" ( + 5 V).  
This is contradictory with for example the RCA evaluations board. Instructions regarding this inputflag are:

Address	Present code	Mnemonic
00BD	3F	BN 4
00C1	37	B 4
00D0	3F	BN 4
00D3	37	B 4
00D9	37	B 4
00F2	3F	BN 4
0A5D	3F	BN 4
0A63	3F	BN 4
0A6A	3F	BN 4
0A77	37	B 4

With this list you can invert the inputflag or use another EF-line. For output Q is used. This is common on nearly all systems; patches are not necessary.

#### B) Cassetteroutines

Included in Tiny Basic are the SAVE and LOAD commands. These make use of two subroutines in the ELF II monitor. Non ELF II owners don't have this monitor and no such subroutines. They can't use SAVE and LOAD. Fortunately a solution is possible.  
Netronics Tiny Basic ends at M 0B87. Here we put the subroutines.

0B80									D3	7B	F8	1D	3B	90	F8	07	1D
0B90	52	FF	01	33	91	39	87	7A	02	30	91	1D	D3	F8	0D	35	
0BA0	9F	35	9B	FF	01	33	A1	3D	A7	30	9C						

Also we must not forget to modify the USER PROGRAM START address at M 0B87.

address	present code	new code
011D	87	AB

The subroutines are called with the SEP register technique. The program counter is RC. This must be initialized with the new starting addresses.

address	present code	new code
09FE	FF	0B
0A01	65	88
0A2A	FF	0B
0A2D	BA	9D

While loading from tape data-bytes are output on the two 7-segment display's with OUT 4. You can change this is necessary.

address	present code	mnemonic
0A4D	64	OUT 4

#### C) Coldstart, Warmstart

Although Netronics Tiny Basic starts at page 01, it is easier to have the cold start at M 0000. For the warm start M 0003 would then be suitable. Here after we can put a simple routine to jump to the systems monitor via a USR Call.

address		opcode	comment
0000	C00100	LBR 0100	cold start
0003	F83D	LDI 3D	delay
0005	BE	PHI RE	
0006	C00103	LBR 0103	warm start
0009	F8C0	LDI C0	monitor high page
000B	B0	PHI R0	
000C	93	GHI R3	= 00
000D	A0	PLO R0	monitor Low = 00
000E	E0	SEX R0	
000F	D0	SEP R0	jump monitor

The instructions at M 0003 and M 0005 could do with some explaining. Included in Netronics Tiny Basic is a software UAR/T. With cold start first the Baudrate is determined (Press CR) and the timing constant is put in R(E)1. Often however the monitor will also use R(E)1 and the timing constant will get lost.

When you use the warm start Tiny assumes that the timing constant in R(E)1 is available and will not again determine the Baudrate. So we load first the delay constant in R(E)1 when using warm start. The value here given (3D) is for 300 BD and a processor clock of 1,75 MHz.

When your systems parameters are different you should look what Tiny has put in R(E)1. (You could use the USR-function for this.)

D) Cassette Loader

When you buy a Netronics Tiny Basic you get a manual and a cassette tape. This saves you about 3000 key-strokes but when the ELF II monitor is not at your disposal you can't read the cassette.

A cassette loader program for ELF II format is listed below.

0000	90	B3	F8	06	A3	D3	93	B2	B7	F8	4E	A2	F8	3E	A7	F8
0010	00	BA	AA	E3	71	23	6E	F8	F9	BD	D7	3B	17	9D	3A	1A
0020	D7	33	20	F8	01	BD	AD	D7	9D	7E	BD	3B	27	D7	8D	F6
0030	C7	7B	00	9D	5A	8A	22	52	67	1A	30	20	1D	D3	F8	0B
0040	35	40	35	3C	FF	01	33	42	3D	48	30	3D	00	00	00	

This program is page relocatable. Put it on a page not written over by Tiny Basic for example 0C00.

There are two startaddresses for R0 or R3 is programcounter. For R0 starting address is 00; for R3 starting address is 02.

Don't forget the page to complete the start address.

Loading starts at M 0000 it continues till the program on tape is finished.

While loading the low address byte is output on the two 7-segment display's.

address	present code
38	67 (OUT 7)

When the program detects a tape error the Q-LED goes on. Because with the Netronics tape format half a cycle is measured it is sensitive to changes in the zero-level. When you get a lot of error messages often it helps to invert the polarity of the tape-signal. In extreme cases you could try experimenting with the timing constant at M 3F. The cassette load program uses EF 2 as input flag and is based on a processorclock of 1,75 MHz.

Enjoy Netronics Tiny Basic!

I'am working for a publishing company in Holland: "De Muiderkring". We have 3 monthly magazines, 2 of them in the electronic field: Elektronica ABC and Radio Bulletin, and one in the field of radio controlled model aeroplanes, boats etc.; HB model & techniek. I'am writing a series in Radio Bulletin on the 1802 micro-processor and can say with some pride that the series is quite successfull. Apart from writing I also developed the computerproject called "Cosmicos". Cosmicos stands for Cosmac micro computersystem. It is based on a small mainboard with 256 bytes RAM, 2 7-segement display's for output and a binary input with pushbuttons and LED's. A selection of expansion boards is also available. All of the expansion boards are based on a common bus, so when you put them in the connector you are ready to go.

At the moment there are 5 expansionboards

- 1) A byte input/output board with AD/DA converter and comparator
- 2) An interface board with eight 7-segment display's hex keyboard interface and cassette interface.

- 3) A 4 K RAM board (8 x 2114 L)
- 4) A 4 K EPROM board (2 x 2716)
- 5) A graphic display board with CDP 1864
- 6) A busboard with 5 connectors.

All these boards are doublesided plated through and the connector-pads are goldplated. Because the boards are small the cost is not prohibitive for hobbyists.

A prototype of a 48 K dynamic RAM board is now running on my own system and the looks of it are very good. Until now it has functioned flawlessly. It is fed with 15 volt unstabilized, has its own stabilisers and DC/DC convertor for the - 5V.

The simplicity of this board is due to the fact that I used the 8202 dynamic RAM controller from Intel.

I think the cost of the board will be in the range \$ 30 - \$ 40 and I will ask my boss if it is possible to make it also available for Canadian and American fellow 1802 users.

When enough people are interested I am willing to write a short article on it with schematics and board lay-out. The dynamic RAM used is 4116.

So long 1802 friends!

H.B. Stuurman  
De Muiderkring b.v.  
P.O. Box 10  
1400 AA Bussum  
Holland

# ELF II SERIAL I/O PACKAGE

-----  
 By: Wes Steiner  
 #1204 2725 Melfa Road,  
 Vancouver, B.C., V6T 1N4  
 -----

If many of you other Elfers out there are frustrated by the difficulty of serial I/O, as I was, then this program will prove to be very helpful. I have an ELF II connected to the Netronics VID and ASCII boards via RS232. For a long time I was restricted to the use of the two hex displays because I didn't know enough about software serial I/O to write my own routines to print and receive characters from the VID and keyboard.

However, thanks to an article in IF #21 by D.JORENS I was able to begin some serious programming. This article presented some short programs which performed the serial I/O. I have adapted them to run with SCRT (ref. RCA 1800 user manual p.61) for greater generality and because this is the technique I use for 99% of my programs.

The program occupies one page of memory with room to spare. The following assumptions must be made before access is allowed to these routines.

- (a) register 2 is the X register and points to a free memory location as the stack.
- (b) register 3 is the program counter.
- (c) register 4,5 point to SCRT CALL and SCRT return respectively.

The program has been assembled starting at 0100h but can be located on any page boundary with the following modifications: assume program is assembled at xy00 then

..all internal calls will be of the form D4 xy ..

..LOCATION CONTENTS

xy31	xy
xy5D	xy
xy22	xy
xyA7	xy

The following are examples of calls to the I/O routines. It is assumed that R4 has the address of SCRTCL( SCRT call routine) and R5 has the address of SCRTRT , R2 points to stack area of memory, and X=2 and R3 is the program counter.

(1) Printing a single character:

```
0507 ..      program code
0508 F8 41      LDI    C'A'
050A BF      PHI    15
050B D4 01 27    SEP    4,A(CPRINT)
050E .. ..      more program code
```

Output:

A\_

(2) Printing a string of immediate characters:

```
0814 ..      program code
0815 D4 01 1D    SEP    4,A(LNEPRT)
0818 54 45 53    C'TEST',#00
081B 54 00
081D .. ..      more program code
```

Output:

TEST\_

(3) Printing an indexed string:

```
0234 49 4E 44 45 58 45 44 20 53 54 52    STR=C'INDEXED STR'
```

```
0E10 ..      program code
0E11 D4 01 94    SEP    4,A(XPRINT)
0E14 02 34 0B    A(STR),L(STR)
0E17 .. ..      more program code
```

Output:

INDEXED STR\_

---

I keep this package on tape and load it in everytime I begin writing a new program. I usually use 00FF for the bottom of the stack, 0100 - 01FF for the I/O routines and my main program begins at 0200h.

Happy Elfing !!

```

0100      ;          S C R T   C A L L
0100      ; THIS IS THE STANDARD CALL AND RETURN TECHNIQUES
0100      ;AS DESCRIBED ON PAGE 61 OF THE RCA 1800 USER
0100      ;MANUAL.
0100
0100 D3      CLLRET SEP      3          ;JUMP TO SUB, P=3 X=2
0101 E2      SCRTCL SEX      2          ;REG 2 IS STACK POINTER
0102 96      GHI      6          ;SAVE
0103 73      STXD          ; RETURN
0104 86      GLO      6          ; ON
0105 73      STXD          ; STACK
0106 93      GHI      3          ;SAVE NEW RETURN IN R6
0107 B6      PHI      6          ;
0108 83      GLO      3          ;
0109 A6      PLO      6          ;
010A 46      LDA      6          ;LOAD ADDR OF CALLEE
010B B3      PHI      3          ;INTO REG 3
010C 46      LDA      6          ;
010D A3      PLO      3          ;
010E 30 00    BR      CLLRET          ;JUMP TO SUB VIA REG 3
0110      ;
0110      ;          S C R T   R E T U R N
0110      ;
0110 D3      RETRET SEP      3          ;
0111 96      SCRTRT GHI      6          ;SET UP R3 FOR RETURN
0112 B3      PHI      3          ;
0113 86      GLO      6          ;
0114 A3      PLO      3          ;
0115 E2      SEX      2          ;R2 IS STACK POINTER
0116 12      INC      2          ;POP THE STACK
0117 72      LDXA          ;SET UP R6 FOR NEXT RET
0118 A6      PLO      6          ;
0119 F0      LDX          ;
011A B6      PHI      6          ;
011B 3010     BR      RETRET          ;JUMP TO CALLER VIA D3
011D      ;          L I N E   P R I N T
011D      ;
011D      ; THIS ROUTINE WILL PRINT THE ASCII EQUIVALENT OF
011D      ;THE BYTES FOLLOWING THE SCRT CALL TO LNEPRT. THE
011D      ;STRING OF BYTES MUST BE TERMINATED BY A 00h.
011D      ;
011D      ; CALLING SEQUENCE: SEP 4,A(LNEPRT)
011D      ;          'ASCII STRING',#00
011D      ;
011D 46      LNEPRT LDA      6          ;GET CHAR TO PRINT
011E BF      PHI      15          ;SAVE IT IN REG 15
011F 32 26    BZ      LPRRET          ;DONE IF DBYTE=00h
0121 D4 01 27 SEP      4,A(CPRINT)    ;PRINT IT
0124 30 1D    BR      LNEPRT          ;GET NEXT CHAR
0126 D5      LPRRET SEP      5          ;RETURN

```

```

0127      ;
0127      ; PRINT
0127      ; THE
0127      ; THIS ROUTINE WILL OUTPUT THE ASCII BYTE IN RF.1
0127      ; USING RS232 SERIAL I/O. THE DELAY TIME IS SET AT
0127      ; 78H. THE CONTENTS OF RF.1 ARE LOST. REGISTERS R7
0127      ; AND R14 ARE SAVED ON ENTRY AND RESTORED ON EXIT.
0127      ;
0127      ; CALLING SEQUENCE: SEP 4,A(CPRINT)
0127      ;
0127 9E      CPRINT GHI      14          ;SAVE R7 ,R14 ON STACK
0128 73          STXD
0129 8E          GLO      14
012A 73          STXD
012B 97          GHI      7
012C 73          STXD
012D 87          GLO      7
012E 73          STXD
012F
012F      ;
012F      ; THE FOLLOWING IS ADAPTED FROM THE ARTICLE IN
012F      ; IPSO FACTO #21 'ELF II SERIAL I/O BY D.J.JORENS'
012F      ;
012F 7A          RESET Q
0130 F8 01      LDI      >A(DELAY)      ;R14 POINTS TO DELAY
0132 BE          PHI      14            ;DELAY IS CALLED BY A
0133 F8 8B      LDI      <A(DELAY)      ;SEP 14.
0135 AE          PLO      14
0136 F8 08      LDI      #08            ;BIT COUNTER
0138 AF          PLO      15
0139 7B          SET      Q
013A DE          SEP      14
013B 9F      PRT010 GHI      15
013C 76          SHRC
013D BF          PHI      15
013E CF          LSDF
013F 7B          SET      Q
0140 38          SKP
0141 7A          RESET Q
0142 DE          SEP      14
0143 2F          DEC      15
0144 8F          GLO      15
0145 3A 3B      BNZ      PRT010
0147 7A          RESET Q
0148 DE          SEP      14
0149 DE          SEP      14
014A 12          INC      2          ;POP THE STACK
014B 72          LDXA
014C A7          PLO      7
014D 72          LDXA
014E B7          PHI      7
014F 72          LDXA
0150 AE          PLO      14
0151 F0          LDXA

```

```

0152 BE          PHI 14
0153 D5          SEP 5          ;RETURN
0154
;
;
;
;
; THIS ROUTINE WILL WAIT FOR A KEY TO BE PRESSED
; ON THE KEYBOARD. THE ASCII CODE WILL BE PUT IN
; RF.1. REGISTERS R7 AND R14 ARE SAVED AND RESTORED.
;
; CALLING SEQUENCE: SEP 4,A(CRECVE)
;
0154 9E          CRECVE GHI 14
0155 73          STXD
0156 8E          GLO 14
0157 73          STXD
0158 97          GHI 7
0159 73          STXD
015A 87          GLO 7
015B 73          STXD
015C F8 01       LDI >A(DELAY)
015E BE          PHI 14
015F F8 8B       LDI <A(DELAY)
0161 AE          PLO 14
0162 F8 00       LDI #00
0164 BF          PHI 15
0165 F8 08       LDI #08
0167 AF          PLO 15
0168 3F 68       RCV010 BN4 RCV010          ;WAIT FOR KEY PRESS
016A F8 3C       LDI #3C
016B 1E 1E       INC 14,INC 14
016E DE          SEP 14
016F 3F 68       BN4 RCV010
0171 DE          RCV020 SEP 14
0172 FC 00       ADI #00
0174 37 78       B4 RCV030
0176 FF 00       SMI #00
0178 9F          RCV030 GHI 15
0179 76          SHRC
017A BF          PHI 15
017B 2F          DEC 15
017C 8F          GLO 15
017D 3A 71       BNZ RCV020
017F DE          SEP 14
0180 12 72       INC 2,LDXA
0182 A7          PLO 7
0183 72 B7       LDXA ,PHI 7
0185 72 AE       LDXA ,PLO 14
0187 F0 BE       LDX ,PHI 14
0189 D5          SEP 5          ;RETURN

```

```

018A      ;
018A      ;
018A      ;
018A      ; THIS ROUTINE PERFORMS THE TIME DELAY FOR THE 300
018A      ; BAUD VID TERMINAL. THE TIME BYTE IS 78H AT 018CH
018A      ; DELAY IS CALLED VIA SEP 14 FROM CPRINT AND CRECVE.
018A      ;
018A D3    DELEXT SEP      3
018B F8 78 DELAY LDI      #78          ; LOAD THE TIME BIT
018D A7      PLO      7          ; INTO R7.0
018E 27      DELO10 DEC      7          ; COUNT DOWN
018F 87      GLO      7          ;
0190 3A 8E      BNZ      DELO10        ;
0192 30 8A      BR       DELEXT        ; RETURN VIA SEP 3
0194      ;
0194      ;
0194      ;
0194      ; THIS ROUTINE WILL PRINT A STRING OF ASCII ENCOD-
0194      ; ED BYTES AT A SPECIFIED ADDRESS IN MEMORY. THE TWO
0194      ; PARAMETERS ARE 2 BYTES FOR THE ADDRESS AND 1 BYTE
0194      ; FOR THE LENGTH OF THE STRING.
0194      ;
0194      ; CALLING SEQUENCE: SEP 4, A(XPRINT)
0194      ;                          A(STRING), L(STRING)
0194      ;
0194 88      XPRINT GLO      8          ; SAVE R8.0, R7
0195 73      STXD
0196 97 73      GHI      7, STXD
0198 87 73      GLO      7, STXD
019A      ; LOAD PARAMETERS INTO R7 AND R8.0
019A 46 B7      LDA      6, PHI 7
019B 46 A7      LDA      6, PLO 7
019C 46 A8      LDA      6, PLO 8
01A0      ; PRINT THE CHARS UNTIL R8.0=0
01A0 88      XPRO10 GLO      8
01A1 32 AB      BZ       XPRO20
01A3 28      DEC      8
01A4 47 BF      LDA      7, PHI 15      ; LOAD CHAR AND SAVE FOR
01A6 D4 01 27    SEP      4, A(CPRINT)  ; OUTPUT BY CPRINT
01A9 30 A0      BR       XPRO10
01AB      ; NOW RESTORE REGISTERS AND RETURN
01AB 12      XPRO20 INC      2          ; POP THE STACK
01AC 72 A7      LDXA     , PLO 7
01AE 72 B7      LDXA     , PHI 7
01B0 F0 A8      LDX      , PLO 8
01B2 D5      SEP      5

```

MEMORY USED : 179 BYTES AT 0100 TO 01B2

Ken Mantei, Chemistry  
 Cal State College  
 San Bernardino, CA 92407

### EPROM PROGRAMMING WITH AN 1802

The cost of a commercial EPROM programmer is high compared with parts costs. Consequently, many experimenters and occasional users may prefer to build their own.<sup>1,2,6</sup> Here is a method originally implemented on a Netronics Elk II 1802 system.<sup>3</sup> It is applicable to any 1802 system that includes RCAs 1853 N-line decoder chip, has functional high address lines, and has some room to add two or three more ICs.

Since most 1802 systems are single voltage, the 2758 (1K bytes), 2716 (2K), and 2732 (4K) 5V EPROM family is a natural choice.<sup>4,5</sup> Each has 24 pins. 20 of these are data, address, and supply voltage which are connected identically throughout the family. Only pins 18-21 vary in function. See Fig. 1.

The key to this programming circuit is the triggering of a 74123 (or similar) monostable multivibrator by an 1802 output instruction to program each byte. A 64 instruction sets the address lines, puts the byte on the data lines, and initiates one flip-flop cycle of the 74123. The  $\overline{Q}_1$  output of the 74123 "one-shot" puts the 1802 into a 50 msec WAIT state freezing the address and data lines.  $Q_1$  or  $\overline{Q}_1$  (depending on the EPROM) also provides the programming pulse. Fig. 2 shows how a 2716 is connected for programming.

### SOFTWARE

Data to be EPROMmed must be loaded into RAM of the 1802 systems in a location such that as a data byte is output by an 1802 instruction, the address lines connected to the EPROM will put the byte in the proper location. For example, to completely program a 2716 that will be used at E000-E7FF that data could be loaded into RAM at 0800-0FFF or 1000-17FF, etc. The programming software is only 33 bytes long. It is keyed in by hand at 0000 so that it executes as soon as the 1802 is put in the RUN mode. See Listing 1. A 64 instruction outputs bytes to a hex LED display on the most systems - change this to 62, 63 or whatever your system uses for display.

HARDWARE

24- and 16-pin wire-wrap sockets should be installed on a kluge board tied into the 1802 system bus. Though not required for EPROM programming, a chip-select signal is needed to read the programmed EPROM. One or two sockets for this might also be installed now. See Fig. 3 for sample  $\overline{CS}$  circuits. Wire on all permanent circuitry shown in Figs 1 and 2. The  $\overline{WAIT}$  signal shown ties into pin 2 of the 1802 regardless of how this pin is labeled on your system (it is called  $\overline{LOAD}$  on the Elf II). The I64 line must come from an 1853 N-line Decoder. Attempts to trigger the 74123 with individual N-lines or N-lines ANDed with TPB caused the WAIT to begin before or after the address lines were all valid. Add an 1853 if your system does not have one.

Knowing that the 1802 WAITS 50 msec for each byte programmed can be used to trim the RC network of the 74123 one-shot. Load in some sample data to be programmed. Key in the programming program. Without applying 25V, time a dry run until the LED output stops changing. It should take about 61 seconds per K bytes to data. One or two 1M resistors added in parallel with the 68K will probably be needed to adjust the RC network.

A 24V programming power supply, as shown in Fig. 4, is a close enough approximation to the 25V specified. Its current is limited to about 25 ma at 24V, a 10 ma margin above the current actually drawn during programming.

Because this author programs infrequently using different EPROMs, and because jumper wires with clips at each end are less expensive and more versatile than switches this admittedly messy approach will be described for connections to the pins 18-21 of the EPROM programming socket. After correct programming of the EPROM has been verified, these pins may be more permanently wired.

## PROCEDURE

1. Load into RAM the data to be EPROMmed (probably from cassette). The starting address must be chosen as mentioned above so the address lines will guide the data into the proper location in EPROM.
2. Customize the programming program, Listing 1, by inserting the starting address of the data in RAM and the number of bytes to be programmed. Key this in at address 0000.
3. Jumper pin 1 of the 74123 to ground.
4. Jumper pins 18-21 as shown in the Table to program the particular EPROM used, connecting the 25V supply last.
5. Flip the RUN switch of the 1802 system on and watch the bytes displayed as they are programmed. Flip off when programming is completed.
6. Disconnect the 25V jumper first, then the rest of the jumpers.
7. Jumper pins 18-21 as shown in the Table to read the particular EPROM used. Verify that all bytes were correctly programmed. I have never experienced failure to a program a few bits but have heard that occasionally it happens, requiring that the programming procedure is repeated.

## A WARNING ABOUT ERASING

Do not try to read an EPROM while erasing it. If it is activated under strong UV light, it gets exceedingly hot and is ruined. This point is not mentioned elsewhere but personal experience has proven it. Pull the EPROM out of its socket and set it into conductive foam for erasure.

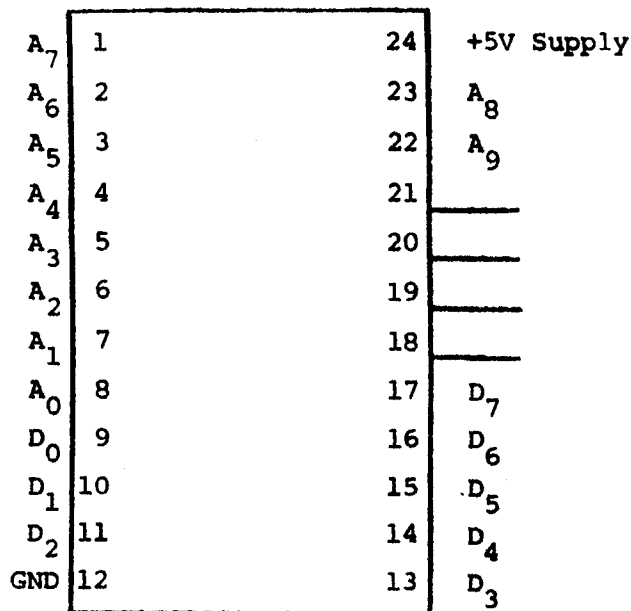
ADDRESSBYTESCOMMENTS

0000	F8 ____ B2F8 ____ A2E2	Point register Z to start of data to be EPROMmed.
0007	F8 ____ B7F8 ____ A7	Load register 7 with the number (in hexadecimal) of bytes to be EPROMmed.
0000	F8 02 B9 F8 72 A9	Provides 10 mses rest between programming pulses when using.
0013	29 99 3A 13	1.79 MHZ clock.
0017	64	Output instruction to triggers one-shot and display bytes.
0018	27 97 3A 00 87 3A 0D	Decrement number of bytes to program out loop until finished
001F	30 1F	STOP

## Listing 1 Programming Program

EPROM References

1. "The 'El Cheapo' EPROM Programmer" Kilobaud (March 1979) p. 46.
2. "1802 EPROM Programming" Kilobaud Microcomputing (March 1980) p. 146.
3. "Expanding the Elf II" Pop. Elec. (March 1978) p. 62.
4. 2716 Spec Sheets by Intel and by Texas Instruments.
5. "E-PROM Doubles Bit Density Without Adding a Pin" Electronics (August 16, 1979) p. 126.  
 Note: The article correctly states that 2716s and 2732s require different polarity for the programming pulses, but has the active high/active low information backwards.
6. "EPROMS and troubleshooting" Kilobaud (sept. 1980) p. 78.



Common Connections for 2758,  
2716, 2732 EPROM Family

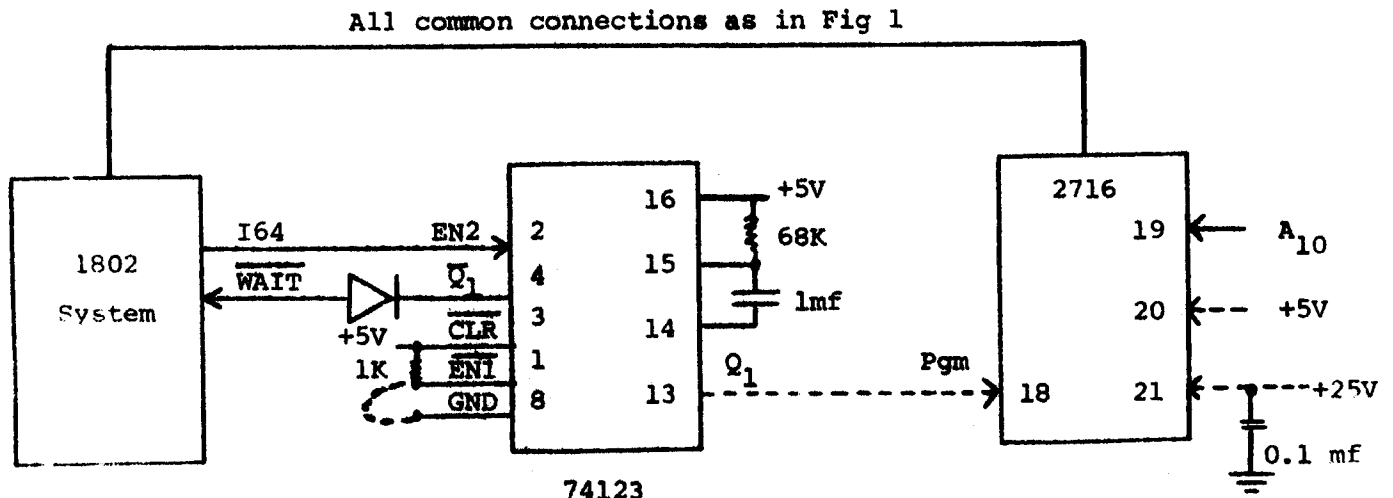
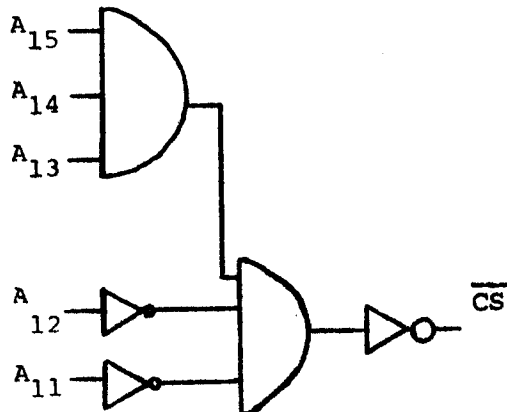
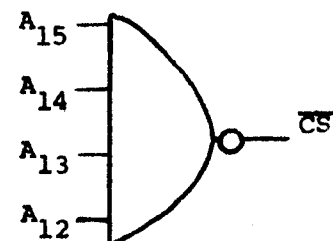


Fig 2

Connections To Program a 2716. Dashed lines indicate temporary jumpers. These are replaced by more permanent connections to  $\overline{CS}$  (18), MR (20), and +5V (21) after programming.



(a)



(b)

Fig3 (a) Chip Select Circuit for 2716 addressed at E000-E7FF using a 4049 and a 4073.

(b) 2732 addressed at F000-FFFF using a 4012.

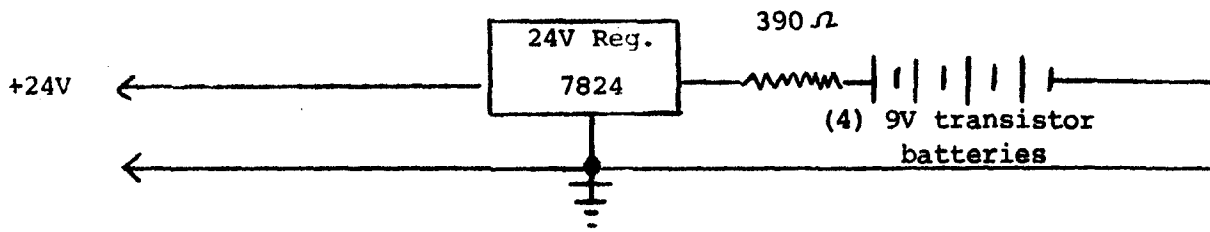


Fig 5. Programming Power Supply

## To Program

pin	2758	2716	2732
21	+25V*	+25V*	A <sub>11</sub>
20	+5V	+5V	+25V*
19	GND**	A <sub>10</sub>	A <sub>10</sub>
18	Q <sub>1</sub>	Q <sub>1</sub>	Q <sub>1</sub>

## To Read

pin	2758	2716	2732
21	+5V	+5V	A <sub>11</sub>
20	MR	MR	MR
19	GND**	A <sub>10</sub>	A <sub>10</sub>
18	CS	CS	CS

Tables showing how jumpers are connected to program and read different EPROMs.

\* Jumper this pin to ground through a 0.1 mf capacitor before connecting the 25V supply to surpress possible transients.

\*\* Some 2758s may require +5V instead.<sup>4</sup>