Ipso Facto

ISSUE 31 OCTOBER 1982

INDEX PAGE

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

1982/1983 EXECUTIVE OF THE ASSOCIATION OF COMPUTER-CHIP EXPERIMENTERS

President:	Tony Hill	416-689-0175
Treasurer:	Ken Bevis	416-277-2495
Directors:	Bernie Murphy Fred Pluthero John Norris Mike Franklin	
Newsletter:		
Production Manager:	Mike Franklin	416-878-0740
Editors:	Fred Feaver Tony Hill	
Advertizing:	Fred Plutero	416-389-4070
Publication:	Dennis Mildon John Hanson	
Hardware & R. and D.:	Don McKenzie Fred Pluthero Ken Bevis Mike Franklin	416-423-7600
Vice-President:	John Norris	416-239-8567
Secretary:	Fred Feaver	416-637-2513
Membership:	Bob Silcox Earle Laycock	416-681-2848
Program Convener:		
Tutorial/Seminars:	Ken Bevis Fred Feaver	
Software:	Wayne Bowdish	416-388-7116
Product Mailing:	Ed Leslie (Publication) Fred Feaver (Boards)	416-528-3222 416-637-2513

CLUB MAILING ADDRESS:

A.C.E.
c/o Mike Franklin
650 Laurier Avenue
Milton, Ontario
Canada
L7T 4R5
416-878-0740

CLUB MEETINGS

Meetings are held on the second Tuesday of each month, September through June at 7:30 in Room B123, Sheridan College, 1430 Trafalgar Road, Oakville, Ontario. A one hour tutorial proceeds each meeting. The college is located approximately 1.0 km north of QEW, on the west side. All members and interested visitors are welcome.

ARTICLE SUBMISSIONS

The majority of the content of Ipso Facto is voluntarily submitted by club members. While we assume no responsibility for errors nor for infringement upon copyright, the Editorial staff verify article content as much as possible. We can always use articles both hardware and software of any level or type relating directly to the 1802 or to micro computer components periferals, products etc. Please specify the equipment or support software upon which the article content applies. Articles which are typed are prefered, and usually printed first, while handwritten articles require some work. Please, please send originals, not photocopy material. We will return photocopies of original material if requested. Photocopies usually will not reproduce clearly.

ADVERTISING POLICY

ACE will accept advertising for commercial products for publication in Ipso Facto at the rate of $25 per quarter page per issue with the advertiser submitting camera-ready copy. All advertisments must be pre-paid.

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 staff will attempt to publish Ipso Facto by the first week of: Issue 31 - Oct. 82, 32 - Dec. 82, 33 - Feb. 83, 34 - Apr. 83, 35 - June 83, and 36 - Aug. 83. 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 6 issues of Ipso Facto published during the club year.

Editor's Corner

Last year at this time, I wrote my first editorial to embark upon the year of developing the "serious system". It was a good year for our members - with the introduction of a quality, high-level language - FORTH - and continued introduction of 1802 hardware - the Dynamic, EPROM and Backplane Boards.

Our President has declared this year, our sixth, to be the year for software development. Work is continuing on developing a quality operating system, and several new or improved high-level languages will be introduced shortly. This issue continues the series on FORTH, and other articles offer useful programs for your use. A point must be made here. You, the user, must make an effort to keep up with the Club in the way it is going. For a start, a minimum of 16K of RAM from 0000H and 4 to 8K of EPROM at C000 or up will be required to use the software and hardware being offered. For members on the ACE buss, our board line offers these features. For ELF II owners - Netronics and John Ware in Texas offer sufficient boards to make this minimum system configuration. Also, the ACE-NAB offers a means of using our boards on the ELF II with a slight reconfiguration of the hardware. Quest owners may also increase their system through Quest products, but not through an ACE/Quest interface (none exists).

BOARD NEWS

VDU Board

In response to increased demand for re-introduction of this board, ACE is offering the VDU for sale on a pre-sale basis until November 10th, 1982. Anyone wishing to add 16K of static RAM or a 1 to 6K video display at address E000 - FFFF, must order this board now with payment in advance (receipts will be issued - use money orders or certified cheques only). The order for the board will be placed on November 15th for the number of boards sold ONLY, with delivery via the post the following week. This sales approach makes a departure from previous sale policy from an existing stock. ACE does not have the financial resources to stockpile an extensive line of boards.

Front Panel

The long-awaited companion to the ACE Backplane is here, in stock, in quantity. The ultimate in user convenience, this board offers a Real Time Clock, an EPROM programmer, IN Port 4 (Hex Pad) and Out Port 4 (Hex Data Leds), a sophisticated Single Step, 16 bit address display via 4 Hex Leds, control circuitry and display, an ACE edge connector for trouble-shooting and a small wire-wrap area. This board connects via two edge connectors to the backplane ver. 1 or 2, and may be connected via ribbon cable to other systems. Price: $35.00.

CPU Board

The ACE CPU Board is functioning beautifully, complete with 1802-4-5-6 compatability, 4-JEDEC EPROM sockets (2-4-8K), 1854 UART, 2 Ports, RS232-C, power-on-reset, selectable Boot, including extensive wire-wrap area. Price: $40.00.

Put away your trainer, or better yet - give it to your kids and move up to a serious micro board, or dedicated controller panel.

SOFTWARE NEWS

Included in this issue is a cassette loader program (Netronics compatible) that will load into your system any cassette distributed by ACE (or Netronics). Starting with the October 12th, 1982 Club Meeting, ACE will make available public domain software to members - bring a cassette and your own recorder.

FORTH is becoming widely used by our membership. Following are a few changes you may wish to make to your listing. To cure a double echo problem from the board (2 characters printed per input) - change:

0A8F H	to	048D
0A77	to	0573
0A79	to	1469

If VLIST bothers you (line over-run, etc.), drop "CR" after each 80 character line. The result is a continuous listing of commands change - 13DF H to 1469.

If your monitor is not located at 8000H, change the following to your own monitor's entry point:

118D H	high byte
1190 H	low byte

Fig FORTH and ACE FORTH

The copy of FORTH distributed by ACE more closely matches COXFORTH than fig FORTH. Fig included an RCA UART driver at address 0543H and disk I/O primitives at address 1503 which was considered to be of little use to the average ACE member. All of the FORTH level words (the dictionary) are identical to fig FORTH. However, beyond address 0575H, the words are advanced 78 bytes. If you buy the fig FORTH Source Listing (highly recommended), you will find the code still useful by calculating the address offset.

The following bibliography on FORTH is recommended to get you into the language:

		USA	CANADA
1. fig FORTH Source Listing for the 1802	fig FORTH	$15.00	$18.00
2. fig FORTH Installation Manual	fig FORTH	15.00	18.00
3. "Starting FORTH", Brodie (paperback)	Bookstore	20.00	20.00
4. "Byte" Reprints on FORTH, 8/80 to 4/81	fig FORTH	5.00	10.00
5. fig Membership and Newsletter	fig FORTH	15.00	27.00

fig: P.O. Box 1105, San Carlos, CA., USA. 94070 - Phone: 415-962-8653

Members' Corner

FOR SALE:

T. Acuff, 1200 - 25th Street, Rock Island, Ill. U.S.A 61201 (309-764-5977)

SUPER-ELF (44 pin buss) - Giant Board, Color/Music Board, Power Supply, Case, RF Modulator, Documentation. Best Offer

HELP:

O. Hoheisel, Herman - bossdorf - str. 33, 2190 Cuxhaven 1, West Germany

Assistance in getting the Quest S100 dynamic board working on an ELF II and Quest Super Expansion Board.

ERRATA:

Mystery Program - I.F. #30, p.25
address (00D0)H = A 9 PLO R9

A CDP 1854 **UART Circuit** - I.F. #29, p.18
Use same inport and outport for 1854, as per schematic - 1 connection required at 1853 to $\overline{\text{CS}}$ of 1854. Alternative: NOR an inport and an outport command together if using different ports.

Note:
$\overline{\text{MRD}}$ on 1854 determines whether a port select is an inport or an outport.

* * * BEST ARTICLE WINNERS 1981/82 * * *

I.F. # 25	-	L.A. Hart	"Kingdom"
I.F. # 26	-	P.B. Liescheski	"Schroedinger Equation"
I.F. # 27	-	J. Swofford	"1802 Real Time Clock"
I.F. # 28	-	M. Franklin	"EPROM Programmer"
I.F. # 29	-	L.A. Hart	"A Bridge Over Troubled Waters"
I.F. # 30	-	T. Hill	"Window"

Congratulations! - and a free year's membership to each of the above.

1802 Computer Conference Report

\- by F. Feaver

The first 1802 Conference of the Association of the Computer-chip Experimenters was held on Saturday, August 7th at the Welland Campus of Niagara College in Welland, Ontario.

In spite of very short preparation time and little advertising, there was still a good turnout of 1802 enthusiasts. Many of those attending were involved in industrial or commercial uses of the 1802 micro, but were not Club Members. They seemed to be getting their money's worth of useful information!

The five speakers gave excellent illustrated talks on the RCA 1802 micro family, its use and application.

The first speaker was our own Mr. Wayne Bowdish, Software Co-ordinator for the Club, who talked on "Writing Quality Software", using handout copies and slides very effectively.

Following Mr. Bowdish was Mr. Ivars Lauzums, Administrator of Marketing and Planning, for RCA Microsystems Division in Somerville, N.J.

Mr. Lauzums told of the development of new RCA 1802 family members such as the 1802A, 1804, 1805, and 1806, some of which contain RAM, ROM, and an enhanced instruction set, which can run at a clock speed of up to 18 MH_z. Mr. Lauzums assured the audience that RCA had not abandoned the users of the 1802 System, but instead was initiating a drive to promote its use with more development systems and high speed chips.

RCA is setting up a software distribution section under the microsystems division which will solicit software from users of the 1802 family and will distribute it on request to others.

Mr. Lauzums gave the Club a new RCA Development System, several high speed 1805 chips and Development Boards as door prizes. Those in attendance were also given four new RCA Manuals.

Mr. Lee Hart, Chief Engineer of Technical MicroSystems Inc. (TMSI) of Ann Arbor, Michigan, was the third speaker. He spoke on the language, FORTH. FORTH is a registered trademark, so TMSI called its revision of FORTH, "EIGHTH".

Mr. Hart outlined the history of FORTH and illustrated how easy it was to program in FORTH using as a subject, a tiny self-propelled Robot "turtle" with a self-contained battery-operated micro. This little Robot was programmed to move so many inches in one direction, stop, play a little tune and change direction. Running on a table top, it never ran over the edge, but instead sensed the lack of secure footing, stopped, played its little ditty and then changed direction, wheeling away from the table's edge.

Cont'd.....

The fourth speaker was Mr. Jan King, Chief Engineer of Amsat, The Amateur Radio Satellite Corporation, charged with the responsibility of launching a satellite into orbit for Radio Amateur communication over a large part of the world. Mr. King told how the FORTH language was used for the radiation hardened 1802 micro controlling the satellite. It was also indicated that the FORTH dictionary had to be bilingual - English and German - to accommodate the two principals: U.S.A. and Germany. There was no redundancy developed into the satellite computer system, which was designed for a ten year life.

The final speaker was one of the founders of the "ACE" Club, Mr. Eugene Tekatch, President of Tektron Equipment Corporation of Stoney Creek, who developed the Tek 1802 microcomputer and introduced the 1802 to Canada. Many hundreds of Tek 1802 computers were sold across Canada and the U.S.A. Mr. Tekatch has successfully applied the 1802 micro to industry in very hostile environments, such as steel mills, in which heat, mechanical shock, vibration and electrical interference would kill most other micros. He has many satisfied customers. Mr. Tekatch discussed some of these problems and shared his experiences with the audience.

He has developed an inexpensive logic probe which has provisions for acting as a pulse injector. Samples of these probe kits as well as other products were given as door prizes.

The Club wishes to thank all the above speakers and also the following electronics companies who kindly donated door prizes: L.A. Varah, White Radio Ltd., AMP of Canada Ltd., Arkon, Tektron Equipment Corp., Western Radio, and RCA.

A fine luncheon and delicious dinner were provided in the Cafeteria of the College.

The Conference Convenors were Bert de Kat and Fred Pluthero, and they did a commendable job. It is considered that the Conference was a success and it is hoped that another can be held next year. Watch for future announcements on the subject!

FORTH IMPLEMENTATION NOTES - II

by - Tony Hill, RR 2 , Hamilton , Ontario , Canada , L8N 2Z7

This article is the second in what I hope will be a continuing series on FORTH for the 1802. I will try to include tips on the 1802 implementation of FORTH as well as some general FORTH tips. I also hope to use these articles as a lead in to other articles about FORTH written by different authors. Please feel free to write.

This month we have an article by Ken Mantei on adding an 1802 assembler to FORTH. Ken is one of the original pioneers in 1802 FORTH and his letters and articles have been most helpful.

1) ERRATTA - FIG-FORTH LISTINGS

Much to our horror, ACE has recently learned that the version of fig-FORTH we have been distributing is not identical to the fig official distribution. There are several copies of 1802 fig-FORTH floating around, and the copy we have been distributing is slightly different. Please note however, that the two versions are FUNCTIONALLY IDENTICAL !

The difference between the two versions is that fig has included UART I/O in the middle of their listing. Unless you have the RCA system this code was written for, the extra bytes only waste memory. Our version does not include them. The fig version also includes disk I/O for the RCA system, which is again useless if you don't have the same disks.

The net result of this difference is that the addresses of the high level fig-FORTH words are offset by a few bytes. Their function and definition has not changed though. Conversely, all the low level (machine code) words are the same in both versions, as the extra UART code is after the rest of the machine code definitions.

Therefore, the comments about patching in I/O published in IPSO FACTO, and the editor's comments about adding Simulated Disk and an Editor in the last issue, apply to the ACE distribution of fig-FORTH. From here on in all articles will be written to apply to both systems, and any differences will be noted. Functionally, you the user will never notice the difference between the two sytems when using FORTH.

2)Bugs in 1802 fig-FORTH

There are a few minor bugs in fig-FORTH. In the next few articles I will discuss some of them, and present some ways to correct them. For example, try asking fig-FORTH whether 20,000 is larger or smaller than -20,000. Or check to see if the computation stack is correctly checked for underflow. Other problems are the effect of an attempt to divide by zero, the way VLIST overflows the edge of the screen and the fact that error numbers are presented in the current base (which you can never remember at the time you get the error). Also annoying is the way CMOVE is defined, such that an attempt to move memory to an overlapping area can mess up the data. While none of these problem are earth shattering, I will discuss some solutions, and will print any other "bugs" that readers send in. Thanks to PVP for pointing out some of the previously mentioned problems.

3)The 1802 Assembler

Included in this issue is an 1802 FORTH assembler. A few notes on using FORTH assemblers are probably in line here, as most of the books I have seen are a little vague on the topic.

First of all, FORTH words written in FORTH ASSEMBLER are usually written in reverse Polish, like most of the rest of FORTH. (Please stop groaning, it's not that bad). Therefore, where you would normally write -

```
GHI RO     or    LDI EA
```

in FORTH ASSEMBLER this would be-

```
0 GHI,     or    EA LDI,
```

Secondly, FORTH words defined in ASSEMBLER are started with the word CODE instead of a " : ". Instead of a " ; ", ASSEMBLER words are terminated with the word NEXT. For example, the following is a FORTH ASSEMBLER word called NOTHING that executes a NOP (C4) instruction -

```
CODE NOTHING NOP, NEXT
```

Another tricky point to remember is that FORTH assemblers do not usually allow labels to identify where branch instructions go to. This is due to the fact that they are not usually two pass assemblers, and thus can not resolve forward references easily. However, as most of the branching in FORTH assembler words is required for loops, a set of assembler level loop words similiar to the high level ones are usually provided.

The loop constructs are

```
   -- IF,   -- ELSE,  -- ENDIF,
     BEGIN,   --   UNTIL,
     BEGIN,   --   AGAIN,
  BEGIN, -- WHILE,  -- REPEAT,
```

and are used the same way as high level loops, except that IF, UNTIL, and WHILE, take the assembler words for the branch instructions as their arguements. For example-

```
CODE  WAIT BEGIN, EF4 UNTIL,  NEXT
```

produces a word WAIT that will wait in a loop on the status of EF4.

Note that you can branch or long branch to an absolute address if you know what it is. For example, the word BYE to exit to a monitor at address 8000 could be written as-

```
CODE BYE  8000 LBR, NEXT
```

Study of the rest of the ASSEMBLER words should prove both educational in the use of the assembler and in the overall power of FORTH as a programming language.

4)Writing Machine Code Words Without An Assembler

It is possible to include hand assembled machine code in FORTH words, with a little bit of work. For example the following is a routine to turn the Q line on by creating a word called QON-

```
HEX CREATE QON 7B C, DC C, SMUDGE
```

Note that a DC op-code must be included in the routine as the last byte (to re-enter the inner interpreter loop). A tip of the hat to Ken Mantei, who first pointed this out to me.

;S

AN 1802 ASSEMBLER FOR 1802 fig-FORTH

by- Ken Mantei,Chemistry Dept.,Cal State College, San Bernardino,Ca. 92407

Once both line and string editing and virtual storage (disc or simulated RAM-disc) have been implemented on an 1802 fig-FORTH system, an ASSEMBLER vocabulary can be developed. Until ASSEMBLER is added, FORTH words are compiled from high level FORTH words. ASSEMBLER allows a FORTH word to be written in machine code. Such words run quicker. More importantly, CODE words can test 1802 flag lines, manipulate the Q line and implement the 6x I/O instructions.

The development and testing of an 1802 fig-FORTH ASSEMBLER has not been completed. What is presented here will certianly be improved on. It will allow one to successfully attach (and patch into FORTH) a first draft ASSEMBLER. It has been used successfully to pop bytes off the FORTH computation stack to an 1802 port, or push them on from a port. The words SEND and READ on lines 8-10 of SCR #8 do this.

To try this out, use the editor to change the error message on SCR #4 line 5 to "OUT OF PAGE BRANCH IN CODE ROUTINE BEING ASSEMBLED". Then enter SCR #6,7,8 UPDATEing after each screen. Type 6 load. If ASSEMBLER is to be a permanent addition to the system type:

```
FORTH DEFINITIONS DECIMAL
LATEST 12 +ORIGIN !
HERE   28 +ORIGIN !
HERE   30 +ORIGIN !
HERE FENCE !
' ASSEMBLER 6 + 32 +ORIGIN !
```

This version for FORTH including compiled page ASSEMBLER may now be saved.

To write a word, called QON, that turns the Q light on type:

```
CODE QON SEQ, NEXT
```

To define HEXKEY?, a word that puts a "1" or "0" on the FORTH stack, depending on the status of 1802 flag 4 type:

```
     CODE HEXKEY? EF4 T/F, NEXT
or   CODE HEXKEY? EF4 NOT T/F, NEXT
```

To move a hex number from the FORTH stack out the 1802 port 4, calling it POP4, type:

```
CODE POP4 4 SEND, NEXT
```

To read input port 4 to the stack type"

```
CODE PUSH4 4 READ, NEXT
```

Notice that ASSEMBLER words generally are followed by a comma. This convention is used to remind one that these words can only be used in CODE definitions. A <BUILD DOES> approach is encountered in the construction of some assemblers, and corrections and improvements to this 1802 fig-FORTH ASSEMBLER are to be expected.

SCR #6 (KAM 30 JULY 80)

```
( FIGFORTH 1802 ASSEMBLER 1)
HEX VOCABULARY ASSEMBLER IMMEDIATE
: ;CODE ?CSP COMPILE (;CODE) [COMPILE] [ SMUDGE
[COMPILE] ASSEMBLER ;  IMMEDIATE
: CODE ?EXEC CREATE [COMPILE] ASSEMBLER !CSP ; IMMEDIATE
ASSEMBLER DEFINITIONS       : STR, F AND 50 OR C, ;
: INC, F AND 10 OR C, ;    : DEC, F AND 20 OR C, ;
: LDN, F AND C, ;          : LDA, F AND 40 OR C, ;
: GLO, F AND 80 OR C, ;    : GHI, F AND 90 OR C, ;
: PLO, F AND A0 OR C, ;    : PHI, F AND B0 OR C, ;
: SEP, F AND D0 OR C, ;    : SEX, F AND E0 OR C, ;
: INP, 7 AND 68 OR C, ;    : OUT, 7 AND 60 OR C, ;
: IDL, 0 C, ;    : NOP, C4 C, ;    : SEP, 7B C, ;   : REQ, 7A C, ;
: SAV, 78 C, ;   : MARK, 79 C, ;   : RET, 70 C, ;   : LDI, F8 C, C, ;
: LDX, F0 C, ;   : LDXA, 72 C, ;   : STXD, 73 C, ; : IRX, 60 C, ;
: XRI, FB C, C, ;  : ORI, F9 C, C, ;  : ANI, FA C, C, ; -->
```

SCR #7 (KAM 30 JULY 80)

```
( FIGFORTH 1802 ASSEMBLER 2)
: DIS, 71 C, ;   : OR, F1 C, ;        : AND, F2 C, ;   : XOR, F3 C, ;
: SHR, F6 C, ;   : SHRC, 76 C, ;      : SHL, FE C, ;   : SHLC, 7E C, ;
: ADD, F4 C, ;   : ADI, FC C, C, ;    : ADC, 74 C, ;   : SD, F5 C, ;
: ADCI, 7C C, C, ;   : SDI, FD C, C, ;   : SDBI, 7D C, C, ;
: SDB, 75 C, ;       : SM, F7 C, ;        : SMI, FF C, C, ;
: SMB, 77 C, ;       : SMBI, 7F C, C, ;
: NEXT C SEP, CURRENT @ CONTEXT ! ?EXEC ?CSP SMUDGE ;
IMMEDIATE    : Q 39 ;     : Z 3A ;     : DF 3B ;   : NOT 8 - ;
: EF1 3C ;   : EF2 3D ;   : EF3 3E ;   : EF4 3F ;
: ?FAULT OVER FF00 AND OVER FF00 AND - 5 ?ERROR ;
: IF, C, HERE 0 C, 2 ;
: ELSE, 2 ?PAIRS 30 C, HERE 1+ SWAP ?FAULT C! HERE 0 C, 2 ;
: ENDIF, ?EXEC 2 ?PAIRS HERE SWAP ?FAULT C! ;
: BEGIN, ?EXEC HERE 1 ;
: UNTIL, C, 1 ?PAIRS HERE ?FAULT DROP C, ;   -->
```

SCR #8 (KAM 30 JULY 80)

```
( FIGFORTH 1802 ASSEMBLER 3)
: BR, HERE 1+ ?FAULT DROP 30 C, C, ;  : LBR, C0 C, , ;
: WHILE, C, 1 ?PAIRS HERE 0 C, 3 ;
: REPEAT, 3 ?PAIRS 30 C, HERE ROT ?FAULT C, 1+ SWAP ?FAULT C! ;
: AGAIN,  1 ?PAIRS 30 C, HERE 1+ ?FAULT DROP C, ;
: 9INC3, 9 INC, 9 INC, 9 INC, ;   : 9DEC3, 9 DEC, 9 DEC, 9 DEC, ;
: T/F, 9INC3, IF, 1 LDI, 9 STR, ELSE, 0 LDI, 9 STR, ENDIF,
  9 DEC, 0 LDI, 9 STR, ;
: READ, INP,  9INC3, 9 STR, 9 DEC, 0 LDI, 9 STR, ;
: SEND, 9 INC, 9 LDN, STXD, IRX, OUT, LDX, STXD,
  9DEC3, ;
;S
```

PROGRAMMING TIPS - LESSON 1

by - Tony Hill and Wayne Bowdish

This article is the first of a series on programming the 1802. It has been a common request from members that we include such a column in each issue, so here's the first ones. We invited comment's and suggestions for what YOU would like to see in future articles.

1)SETTING UP A PROGRAM COUNTER

Almost all 1802 programs are written to use R(3) as the program counter. However, the 1802 initializes with R(0) as for a program counter. To make matters worse, the execute command of many monitors sets R(3) as the program counter before running your program. What follows here is a piece of relocatable code you can place at the start of your program that will allow entry with either R(0) or R(3) as the program counter. With this code at the start, you can no longer have to specify whether register 0 or 3 is the PC when a progrm is run.

```
00 F8 06        LDI     START   ; LOAD LOW BYTE OF PROGRAM START LOCATION
02 A3           PLO     R3      ; PUT IT INTO R(3).0 (JUMP IF R(3) IS PC)
03 90           GHI     RO      ; GET HIGH BYTE OF R(0) IF IT IS THE PC
04 B3           PHI     R3      ; AND PUT IT IN R(3).1
05 D3           SEP     R3      ; SET R(3) AS THE NEW PC
06 ..   START:  ...             ; AND CONTINUE WITH THE REST OF THE PROGRAM
```

This code can be modified to work with any entry register, at the expense of the code being non-relocatable-

```
00 F8 07        LDI     START       ; LOAD LOW BYTE OF PROGRAM START LOCATION
02 A3           PLO     R3          ; PUT IT INTO R(3).0 (JUMP IF R(3) IS PC)
03 F8 xx        LDI     START/256   ; GET HIGH BYTE OF THE START ADDRESS
05 B3           PHI     R3          ; AND PUT IT IN R(3).1
06 D3           SEP     R3          ; SET R(3) AS THE NEW PC
07 ..   START:  ...                 ; AND CONTINUE WITH THE REST OF THE PROGRAM
```

2) SIMULATED STACK INSTRUCTIONS

The 1802 has a number of instructions for handling a stack. These include LDX , LDXA , STXD , IRX , INP and OUT. However, a number of useful stack manipulation instructions found on other micro's are missing. This program tip is designed to illustrate how to "fake" these instructions when you are writing code that does not know for sure which register is the stack pointer.

PSEUDO INSTRUCTION	REQUIRED 1802 OP-CODES	FUNCTION
DEX	LDX , STXD	Decrement the stack pointer
POP	IRX , LDX	Load D with top byte on stack
STX	STXD , IRX	Store D on top of stack

There may be other useful pseudo instructions (stack or otherwise), and we will be happy to print any that are sent in. Note that the DEX instruction destroys the D accumulator contents

NEXT ISSUE -

In the next IPSO, we will talk about LOOPs, and explain various way to implement them, including a discussion on nested loops. Also planned for future issues are tips on data structures, I/O programming and other nonsense.

ADDITIONAL NOTES ABOUT THE WINDOW PROGRAM

by- Tony Hill , RR 2 , Hamilton , Ontario , Canada , L8N 2Z7

In response to inquires about my WINDOW program (I.F. #24) I have listed a number of additional comments below, most of which should probably have been in the original article. I also neglected to credit Wayne Bowdish, whose dissassembler code I modified for use in the program.

ADDITIONAL INTERESTING MEMORY LOCATIONS

02C4	69	An "INP 1" instruction used to read the keyboard in SWAP mode. Used to reset the keyboard when swapping the WINDOW display back in.
0028	E0	Video display RAM high order address byte
0030	E3	Video display RAM high order address byte + 3
003B	E2	Video display RAM high order address byte + 2
0091	E1	Video display RAM high order address byte + 1
00E2	E0	Video display RAM high order address byte
01B7	E2	Video display RAM high order address byte + 2
01C3	E2	Video display RAM high order address byte + 2
01C7	E0	Video display RAM high order address byte
0357	E0	Video display RAM high order address byte
0573	70	Video display RAM high order address byte shifted right
0588	E0	Video display RAM high order address byte
05C6	E0	Video display RAM high order address byte
0614	E1	Video display RAM high order address byte + 1
062E	E0	Video display RAM high order address byte

REGISTER USAGE

R0	not used	R8	"2 byte subroutine call" return
R1	"	R9	RAM page pointer
R2	stack pointer (grow down)	RA	General memory pointer
R3	program counter	RB	Video RAM pointer
R4	SCRT call	RC	Op-code high/low nibble storage
R5	SCRT return	RD	general purpose register
R6	SCRT address storage	RE	"
R7	"2 byte subroutine call" call	RF	RF.1 passes D for SCRT

INPUT INSTRUCTIONS AND FLAG LINES

When WINDOW finds an input instruction (or a branch on flag condition) it stops whatever mode it was in and asks for the HEX value to use as the required input data (or the status of the flag line - 0 or 1). You simply type in your answer and press a carriage return to continue in whatever mode you were in when the instruction was found.

ADDITIONAL NOTE FOR NON 6847 SYSTEMS

As a result of the number of inquires I have received about using WINDOW on non-6847 display systems, I am currently writing a version that will run on any display capable of accepting the printable ASCII characters and CR/LF. While I will have to eliminate all graphics and make the output format simple, the basic functionality will be the same. In view of the fact that the new program will lack the exciting graphics of the old one, I am going to name the new program "PEEPHOLE". Watch for it around the end of the year.

PERMUTATIONS AND COMBINATIONS IN TINY BASIC

by - K Schoedel , RR #1 , Erin , Ontario , Canada , NOB 1TO

Everyone knows that Tiny Basic's math capability is severely limited. Only integers can be used, and not very large ones at that. It is not even practical to use Tiny Basic for many day to day calculations. Obviously, any form of higher math is impossible.

Not so. There are many potentially useful operations that use only relatively small integers. Permutations and combinations fall into this category. These useful formulas can help you answer many pressing everyday questions, like "How many ways can I arrange the 4116's on my 64K board?".

The permutation formula provides the number of ways that **r** objects taken from a set of **n** can be arranged. The standard formula for this is:

$$P = \frac{n!}{(n-r)!}$$

where **n** is the total number of objects and **r** is the number to be chosen and arranged

The main problem with calculating permutations in TINY BASIC is that factorials are used. The factorial of a number, represented by an exclamation mark (!) is equal to all of the integers from one to that number multiplied together. This quickly yields very large numbers; even 8! is outside Tiny's normal number range. However, it is not necessary to calculate the entire factorial to do permutations. For example, in taking three objects from a group of ten we get:

$$P = \frac{10!}{(10-3)!} = \frac{10*9*8*7*6*5*4*3*2*1}{7*6*5*4*3*2*1} = 10*9*8 = 720$$

Since (7*6*5...*1) appears in both the numerator and the denominator it can be cancelled out and need not be calculated. This allows P(10,3) to be calculated even though 10! is far outside Tiny Basic's number range.

The formula for combinations is very similiar. It is:

$$C(n,r) = \frac{n!}{r!(n-r)!}$$

This is very similiar to the formula used for calculating permutations. The only difference is the extra r! in the denominator. The number of combinations is therefore the same as the number of permutations divided by the factorial of **r**. The program could in fact calculate combinations this way, but it does not. Doing so would place a severe restriction on the range of acceptable values. Instead, the program does the division by r! piece by piece in between multiplications. This increases by several times the number of values that the combinations program can calculate.

Here are a few examples showing the use of this program.

```
:RUN
PERMUTATIONS OR COMBINATIONS?  P
TOTAL NUMBER?  5
TAKE?  3
3 ITEMS TAKEN FROM 5
AND ARRANGED ON 60 WAYS.
```

(How any ways can you arrange 3 of your 5 years of IPSO on a bookshelf?)

```
PERMUTATIONS OR COMBINATIONS? C
TOTAL NUMBER? 20
TAKE? 3
3 ITEMS CAN BE TAKEN FROM 20
IN 1140 WAYS.
```

(How many ways can a committe of 3 people be chosen from 20 members?)

So, Tiny Basic isn't quite as useless with numbers as it is always made out to be. Just because "Tiny can't handle things like that" is no reason to ignore it; with suitable programs it really can be quite powerful.

PERMUTATIONS AND COMBINATIONS IN QUEST TINY BASIC V3.0

```
 10 PR
 20 PR "PERMUTATIONS OR COMBINATIONS" ;
 30 P=0
 40 C=1
 50 INPUT T
 60 PR "TOTAL NUMBER" ;
 70 INPUT N
 80 M=N
 90 PR "TAKE" ;
100 INPUT R
110 S=R
120 F=1
130 F=F*M
140 IF F<0 GOTO 290
150 M=M-1
160 IF T=C GOSUB 240
170 IF M>N-R GOTO 290
180 IF S>1 IF T=C GOSUB 240
185 IF S>1 IF T=C GOTO180
190 PR R ; " ITEMS CAN BE TAKEN FROM " ; N
200 IF T=P PR "AND ARRANGED " ;
210 PR "IN " ; F ; "WAYS."
220 GOTO 10
230 END
240 IF F/S*S<>F RETURN
250 F=F/S
260 S=S-1
270 IF S=0 S=1
280 RETURN
290 PR "SORRY... THAT'S TOO LARGE"
300 END
```

SOFTWARE FOR THE ACE VDU BOARD

by - Tony Hill, RR 2 , Hamilton , Ontario , Canada , L8N 2Z7

A number of article containing software for memory mapped video displays, including 6847 based units, have appeared in IPSO FACTO over the last few years. However, a few club members have written in to complain that there was nothing written specifically for the ACE VDU board. As a result, this article has been written to provide a simple routine to allow the VDU board to be used as an alpha-numeric output device.

First of all, I should say that I have taken the code for this routine almost directly from the video output routine in NIES MONITOR - VERSION II and so credit goes to Steve Nies as the original author. I have made some additions and modifications to convert his routine to be a stand alone subroutine.

The software consists of a single SCRT callable subroutine that can be placed at the start of any page in memory (RAM or PROM). It assumes that SCRT passes the value you left in the D accumulator in R(F).1 . To use the code, simply CALL it at its first address with the character you want to display in D. What could be easier?

A few other notes are in order. The code will clear the screen, initialize the cursor AND SET THE SCREEN TO ALPHA MODE when you pass it a form-feed character (0C HEX). Therefore, the first thing any program should do is send a form-feed to initialize the display.

The routine will print all MC6847 ASCII characters, and also handle the following ASCII control codes-

```
BS (08)  - backspace
CR (0D)  - carriage return
LF (0A)  - line feed
FF (0C)  - form feed
HT (09)  - tab (move right to next column of 8)
VT (0B)  - vertical tab (move the cursor up one line)
-> (12)  - right arrow (DC2) (move cursor right one column)
```

Other control codes are ignored.

The routine saves all registers that it uses. Therefore, it will not conflict with the register assignmment of any program it is used with. In addition, the code passes back the same character passed to it, which makes it available for further processing.

A simple example program, to print the letter "A" in the upper left hand corner of the screen might be-

```
0000    F8 0C       LDI    #0C      ; SET UP THE SCREEN WITH A FORMFEED
0002    D4 xx00     +CALL  VDUOUT   ; SEND TO VDU OUTPUT ROUTINE
0005    F8 41       LDI    #41      ; LOAD THE ASCII CHARACTER "A"
0007    D4 xx00     +CALL  VDUOUT   ; SEND TO VDU OUTPUT ROUTINE
000A    00          IDL             ; END
```

The final point you need to consider is that the code requires 2 bytes of RAM (anywhere in memory) to store the current cursor position in. The code as presented here is written to use the two bytes in the third page of the video memory 2114's as the storage location. While this memory is not used by the 6847, if you wish to use it for anything else, you will have to modify the code accordingly (the values labeled CURSAV).

```
                         ;*************************************************************
                         ;*   ACE VDU BOARD OUTPUT ROUTINES                           *
                         ;*         FROM CODE WRITTEN BY - STEVE NIES                 *
                         ;*              MODIFIED BY     - TONY HILL                  *
                         ;*************************************************************
           =   E0 00     VDU:     .EQL     #E000          ; START ADDRESS OF VIDEO RAM
           =   E3 00     CURSAV:  .EQL     #E300          ; CURSOR POSITION SAVE LOACATION
           =   FF 00     CTLREG:  .EQL     #FF00          ; VIDEO MODE CONTROLL ADDRESS

0000       98            VDUOUT:  GHI      R8             ; SAVE R(8)
0001       73                     STXD                    ;
0002       88                     GLO      R8             ;
0003       73                     STXD                    ;
0004       97                     GHI      R7             ; SAVE R(7)
0005       73                     STXD                    ;
0006       87                     GLO      R7             ;
0007       73                     STXD                    ;
0008       9F                     GHI      RF             ; CHECK FOR A FORMFEED CHARACTER
0009       FF 0C                  SMI      #0C            ;
000B       32 BE                  BZ       FF             ; PROCESS IMMEDIATLY IF FOUND
000D       F8 E3                  LDI      CURSAV/256     ; R(7) -> CURSOR SAVE LOCATION
000F       B7                     PHI      R7             ;
0010       F8 00                  LDI      CURSAV         ;
0012       A7                     PLO      R7             ;
0013       47                     LDA      R7             ; SET R(8) -> CURSOR POSITION
0014       B8                     PHI      R8             ;
0015       47                     LDA      R7             ;
0016       A8                     PLO      R8             ;
0017       08                     LDN      R8             ; TURN THE CURSOR OFF
0018       FA 7F                  ANI      #7F            ; MASK OFF INVERT BIT
001A       58                     STR      R8             ;
001B       9F            SKIP:    GHI      RF             ; GET CHARACTER PASSED BY SCRT
001C       FF 20                  SMI      #20            ; TEST FOR A CONTROL CHARACTER
001E       3B 64                  BNF      CNTL           ; BRANCH IF IT IS ONE
0020       9F                     GHI      RF             ; OTHERWISE, GET IT AGAIN
0021       FA BF                  ANI      #BF            ; MASK OF GRAPHICS BIT 6
0023       58                     STR      R8             ; SAVE ON SCREEN
0024       18            NEXT:    INC      R8             ; MOVE THE CURSOR LEFT ONE SPACE
0025       98            TEST:    GHI      R8             ; SCROLL SCREEN UP ONE LINE ?
0026       FF E2                  SMI      VDU/256+2      ;
0028       3B 4A                  BNF      FINISH         ; BRANCH IF NOT
002A       88                     GLO      R8             ; OTHERWISE SAVE CURRENT CURSOR POSITION
002B       52                     STR      R2             ;
002C       F8 E0                  LDI      VDU/256        ; SET UP R(7) AND R(8) TO DO THE SCROLL
002E       B7                     PHI      R7             ;
002F       B8                     PHI      R8             ;
0030       F8 00                  LDI      VDU            ;
0032       A8                     PLO      R8             ;
0033       F8 20                  LDI      VDU + #20      ;
0035       A7                     PLO      R7             ;
0036       47            SCROLL:  LDA      R7             ; MOVE THE SCREEN UP A LINE
0037       58                     STR      R8             ;
0038       18                     INC      R8             ;
0039       97                     GHI      R7             ;
003A       FF E2                  SMI      VDU/256+2      ;
003C       3B 36                  BNF      SCROLL         ;
003E       F8 20         BLANK:   LDI      #20            ; BLANK OUT BOTTOM LINE
0040       58                     STR      R8             ;
0041       18                     INC      R8             ;
0042       88                     GLO      R8             ;
0043       3A 3E                  BNZ      BLANK          ;
0045       28                     DEC      R8             ;
0046       02                     LDN      R2             ; RESTORE THE CURRENT CURSOR POSITION
0047       FF 20                  SMI      #20            ;
0049       A8                     PLO      R8             ;
004A       08            FINISH:  LDN      R8             ; TURN THE CURSOR ON
004B       F9 80                  ORI      #80            ;
004D       58                     STR      R8             ;
004E       F8 E3                  LDI      CURSAV/256     ; SET R(7) -> CURSOR SAVE POSITION
0050       B7                     PHI      R7             ;
0051       F8 00                  LDI      CURSAV         ;
0053       A7                     PLO      R7             ;
0054       98                     GHI      R8             ; SAVE CURSOR POSITION
0055       57                     STR      R7             ;
0056       17                     INC      R7             ;
0057       88                     GLO      R8             ;
0058       57                     STR      R7             ;
0059       12                     INC      R2             ; RESTORE R(7)
005A       42                     LDA      R2             ;
005B       A7                     PLO      R7             ;
005C       42                     LDA      R2             ;
```

```
005D  B7              PHI     R7            ;
005E  42              LDA     R2            ; RESTORE R(8)
005F  A8              PLO     R8            ;
0060  02              LDN     R2            ;
0061  B8              PHI     R8            ;
0062  9F              GHI     RF            ; RESTORE PASSED CHARACTER
0063  D5              +RETRN                ;
                ;
                ;
0064  FB ED     CNTL: XRI     #ED           ; TEST FOR CONTROL CHARACTERS
0066  3A 6E           BNZ     LF            ; ** CARRIAGE RETURN? **
0068  88              GLO     R8            ; YES, SO MOVE CURSOR TO START OF LINE
0069  FA E0           ANI     #E0           ;
006B  A8              PLO     R8            ;
006C  30 4A           BR      FINISH        ;
006E  FB 07     LF:   XRI     #07           ;
0070  3A 7C           BNZ     BS            ; ** LINE FEED ? **
0072  88              GLO     R8            ; YES, MOVE CURSOR DOWN ONE LINE
0073  FC 20           ADI     #20           ;
0075  A8              PLO     R8            ;
0076  98              GHI     R8            ;
0077  7C 00           ADCI    00            ;
0079  B8              PHI     R8            ;
007A  30 25           BR      TEST          ; GO SEE IF WE NEED TO SCROLL
007C  FB 02     BS:   XRI     #02           ;
007E  3A 88           BNZ     HT            ; ** BACKSPACE ? **
0080  28              DEC     R8            ; YES, MOVE CURSOR BACK AND TEST FOR LIMIT
0081  98        EXIT: GHI     R8            ;
0082  FF E0           SMI     VDU/256       ;
0084  3B 24           BNF     NEXT          ;
0086  30 25           BR      TEST          ;
0088  FB 01     HT:   XRI     #01           ;
008A  3A 98           BNZ     VT            ; ** HORIZONTAL TAB ? **
008C  88              GLO     R8            ; YES -TAB OVER TO THE NEXT COLUMN
008D  FC 08           ADI     #08           ;
008F  FA F8           ANI     #F8           ;
0091  A8              PLO     R8            ;
0092  98              GHI     R8            ;
0093  7C 00           ADCI    #00           ;
0095  B8              PHI     R8            ;
0096  30 25           BR      TEST          ; CHECK TO SEE IF WE NEED TO SCROLL
0098  FB 02     VT:   XRI     #02           ;
009A  3A B0           BNZ     DC2           ; ** VERTICAL TAB (0B) ? **
009C  88              GLO     R8            ; IGNORE IT IF ON LINE 1
009D  FF 20           SMI     #20           ;
009F  33 A6           BDF     UP            ;
00A1  98              GHI     R8            ;
00A2  FF E1           SMI     #E1           ;
00A4  3B 4A           BNF     FINISH        ;
00A6  88        UP:   GLO     R8            ; OTHERWISE MOVE UP ONE LINE
00A7  FF 20           SMI     #20           ;
00A9  A8              PLO     R8            ;
00AA  98              GHI     R8            ;
00AB  7F 00           SMBI    #00           ;
00AD  B8              PHI     R8            ;
00AE  30 4A           BR      FINISH        ;
00B0  FB 19     DC2:  XRI     #19           ;
00B2  32 24           BZ      NEXT          ; ** RIGHT ARROW (12,DC2) ? **
00B4  FB 16           XRI     #16           ; ** HOME CURSOR ? (04) **
00B6  3A 81           BNZ     EXIT          ; IGNORE IF NOT-INVALID CONTROL CODE
00B8  A8              PLO     R8            ; OTHERWISE POINT CURSOR TO UPPER LEFT CORNER
00B9  F8 E0           LDI     VDU/256       ;
00BB  B8              PHI     R8            ;
00BC  30 4A           BR      FINISH        ;
                ;
00BE  F8 E1     FF:   LDI     VDU/256 +1    ; ** FORM FEED **
00C0  B8              PHI     R8            ;
00C1  F8 FF           LDI     #FF           ;
00C3  A8              PLO     R8            ;
00C4  F8 FF           LDI     CTLREG/256    ; SET SCREEN TO ALPHA MODE
00C6  B7              PHI     R7            ;
00C7  F8 02           LDI     #02           ;
00C9  57              STR     R7            ;
00CA  F8 20     CLEAR: LDI    #20           ; CLEAR VIDEO MEMORY
00CC  58              STR     R8            ;
00CD  28              DEC     R8            ;
00CE  98              GHI     R8            ;
00CF  FF E0           SMI     VDU/256       ;
00D1  33 CA           BDF     CLEAR         ;
00D3  30 24           BR      NEXT          ; RESTORE CURSOR
                      .END
```

1802-Apple Keyboard Interface

-by J. Pottinger, 505 E. Lakeside Dr., Florence, Ala. U.S.A.,35630

Are you a home brew hacker? Do you have a hex key pad with keys that bounce when a car pulls into the drive? Then you might use the simple fix presented here. The hex key pad and circuit were presented by Thomas E. Hutchinson KILOBAUD, November 1978.

The parts list for this project is pretty short. You will need a 555 timer (wired as a one shot-see Figure 1), a 16 pin double ended DIP jumper cable and a solderless breadboard or equivalent. Oh yes, don't forget the APPLE. Unfortunately they don't grow on trees. I use the APPLE II PLUS with one disk drive.

Interfacing is simple, just remove the four gates (IC's number 1, 2, 3 and 4) used to decode the key pad and wire pin 13 of each gate socket to the appropriate annunciator on the APPLE game connector (see Figure 1 AN 0 to AN 3). IC 10 can also be removed. The strobe from the APPLE is slowed by the one shot that replaces the fifth gate of Hutchinson's circuit. This still uses the key debouncer which isn't necessary, but seemed to be the shortest route to success.

A software driver for this circuit is shown in the following listing. It is written in Applesoft basic. The program makes a hex key pad from the right side of the standard APPLE key board. The program accepts a nibble at a time from the keyboard and places them in APPLE memory and the registers of the hex input circuit on the ELF. A simple subroutine should allow programs entered this way to be saved to disk and stored for later down loading to the ELF.

One problem with this circuit has surfaced. The debounce circuit limits the speed of transfer. There is no doubt a simple fix, but I haven't had the opportunity to try anything yet. Maybe someone is ACE with simular interests can enchance this simple circuit or maybe a simple compiler for one page ELFs or maybe...

```
1  GOTO 1000: REM   SUBROUTINES FOR INTERPRETING KEYPAD FOLLOW
2  PRINT "ERROR, REPEAT ENTRY": GOTO 300
3  GOTO 2000
17  END
22  PRINT " ";: POP : GOTO 300
34 A = 2:B$ = "2": GOSUB 5080: RETURN
36 A = 3:B$ = "3": GOSUB 5080: GOSUB 5140: RETURN
44 A = 12:B$ = "C": GOSUB 5100: GOSUB 5120: RETURN
45 A = 13:B$ = "D": GOSUB 5100: GOSUB 5120: GOSUB 5140: RETURN
46 A = 14:B$ = "E": GOSUB 5100: GOSUB 5120: GOSUB 5080: RETURN
47 A = 15:B$ = "F": GOSUB 5100: GOSUB 5120: GOSUB 5080: GOSUB 5140: RET
URN
62 A = 4:B$ = "4": GOSUB 5100: RETURN
63 A = 10:B$ = "A": GOSUB 5120: GOSUB 5080: RETURN
64 A = 5:B$ = "5": GOSUB 5140: GOSUB 5100: RETURN
65 A = 6:B$ = "6": GOSUB 5100: GOSUB 5080: RETURN
66 A = 7:B$ = "7": GOSUB 5100: GOSUB 5080: GOSUB 5140: RETURN
67 A = 1:B$ = "1": GOSUB 5140: RETURN
68 A = 0:B$ = "0": RETURN
69 A = 11:B$ = "B": GOSUB 5120: GOSUB 5080: GOSUB 5140: RETURN
75 A = 9:B$ = "9": GOSUB 5120: GOSUB 5140: RETURN
79 A = 8:B$ = "8": GOSUB 5120: RETURN
```

BY J M POTTINGER
QCAB
C/O ANDERSON COMPUTERS
FLORENCE ,AL   35630

```
110  REM
115  REM
120  REM   HEX KEY PAD
140  REM   USING KEYS
150  REM   ************
160  REM   *6 7 8 9   *
170  REM   * Y U I O  *
180  REM   *  H J K L *
190  REM   *   N M , .*
200  REM   ************
210  REM        AS
220  REM   ************
230  REM   *C D E F   *
240  REM   * 8 9 A B  *
250  REM   *  4 5 6 7 *
260  REM   *   0 1 2 3*
270  REM   ************
289  REM   CONVERTS KEYS TO HEX
290  ON  ASC (A$) - 10 GOSUB 2,2,3,2,2,2,2,2,2,2,2,2,2,2,2,2,17,2,2,2,2
,22,2,2,2,2,2,2,2,2,2,2,2,34,2,36,2,2,2,2,2,2,2,44,45,46,47,2,2,2,2,2,2
,2,2,2,2,2,2,2,2,62,63,64,65,66,67,68,69,2,2,2,2,2,75,2,2,2,79
291  RETURN
299  REM   MAIN ROUTINE
300  GET A$
310  GOSUB 290: GOSUB 500
320 B1 = A * 16
325  PRINT B$;
330  GET A$: GOSUB 290: GOSUB 500
340 B2 = A
350 B = B1 + B2
355  PRINT B$;
360  POKE ADR,B
370 ADR = ADR + 1
380 ICT = ICT + 1
390  IF ICT = 8 THEN  GOTO 2000
400  PRINT " ";: GOTO 300
500  GOSUB 5060: REM  STROBE OUTPUTS
510  FOR I = 1 TO 200
520  NEXT I
530  GOSUB 5020: REM  CLEAR ALL OUTPUTS
540  RETURN
1000  GOSUB 5200: GOSUB 3000: REM INIT SUB FOR HEX OUTPUT
1005  INPUT "INPUT THE STARTING ADDRESS ";ADR
1010  HOME
1040  GOTO 2000
1999  REM   POKES ADDRESS IN ROUTINE FOR HEX OUTPUT
2000  PRINT : POKE 768,ADR / 256
2010  POKE 769,(ADR -  PEEK (768) * 256)
2020  CALL 770: REM   HEX OUTPUT ROUTINE
2030  PRINT " ";
2040 ICT = 0
2050  GOTO 300
3000  FOR I = 1 TO 10
3010  READ D
3020  POKE 769 + I,D
```

```
3030  NEXT I
3035  PRINT "THE END OF ARRAY STORAGE IS ";
3036  PRINT  PEEK (110) * 256 +  PEEK (109)
3040  RETURN
3050  DATA   173,0,3,174,1,3,32,65,249,96
5010  REM  CLEAR ALL OUTPUTS
5020  FOR I = 0 TO 3
5030  POKE LAN(I),0
5040  NEXT I
5050  RETURN
5060 X =  PEEK (STROBE)
5070  RETURN
5080  POKE HAN(1),0: REM  BIT 1= 2
5090  RETURN
5100  POKE HAN(2),0: REM  BIT 2= 4
5110  RETURN
5120  POKE HAN(3),0: REM  BIT 3= 8
5130  RETURN
5140  POKE HAN(0),0: REM  BIT 0= 1
5150  RETURN
5160  POKE LAN(0),0: RETURN
5170  POKE LAN(1),0: RETURN
5180  POKE LAN(2),0: RETURN
5190  POKE LAN(3),0: RETURN
5200 STROBE =  - 16320
5210  DIM LAN(3),HAN(3)
5220 LAN(0) =  - 16296
5230 HAN(0) =  - 16295
5240 LAN(1) =  - 16294
5250 HAN(1) =  - 16293
5260 LAN(2) =  - 16292
5270 HAN(2) =  - 16291
5280 LAN(3) =  - 16290
5290 HAN(3) =  - 16289
5295  RETURN
5300  GOSUB 5010
5310  GET A$
5320  IF A$ = "S" THEN  GOSUB 5060
5330  IF A$ = "4" THEN  GOSUB 5160
5340  IF A$ = "1" THEN  GOSUB 5080
5350  IF A$ = "5" THEN  GOSUB 5170
5360  IF A$ = "2" THEN  GOSUB 5100
5370  IF A$ = "6" THEN  GOSUB 5180
5380  IF A$ = "3" THEN  GOSUB 5120
5390  IF A$ = "7" THEN  GOSUB 5190
5400  IF A$ = "0" THEN  GOSUB 5140
5410  IF A$ =  CHR$ (3) THEN  END
5420  GOTO 5310
5430  GOSUB 5010
5440  FOR I = 0 TO 3
5450  POKE LAN(I),0
5460  GET A$
5470  POKE HAN(I),0
5480  GET A$
5490  NEXT I

]
```

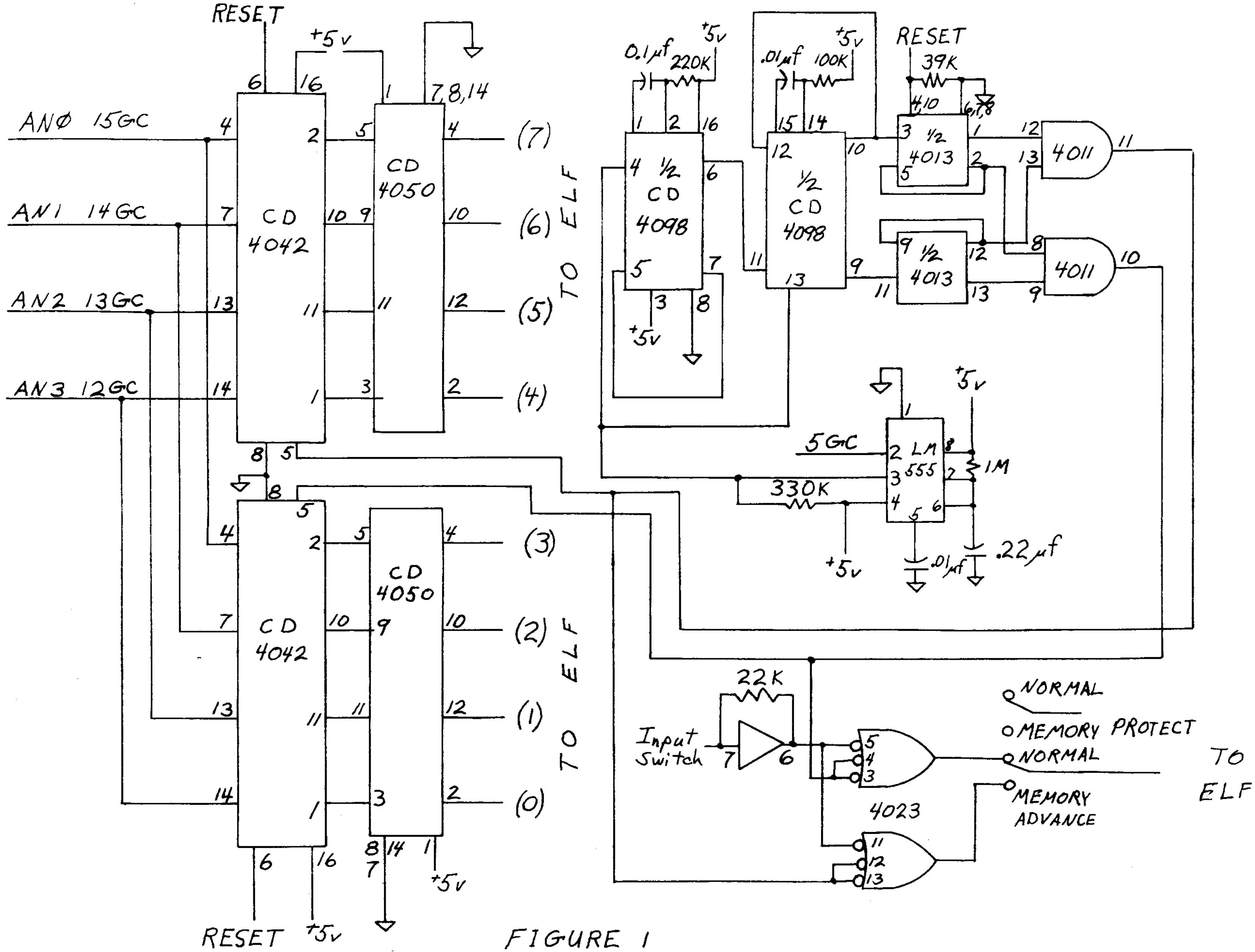

RESET
+5v
AN∅ 15GC
AN1 14GC
AN2 13GC
AN3 12GC
CD 4042
CD 4050
7,8,14
(7)
(6)
(5)
(4)
(3)
(2)
(1)
(0)
TO ELF
0.1μf
220K
.01μf
100K
½ CD 4098
39K
½ 4013
4011
LM 555
5GC
330K
1M
.22μf
.01μf
22K
Input Switch
4023
NORMAL
MEMORY PROTECT
MEMORY ADVANCE
TO ELF

FIGURE 1

1802 mini-DISASSEMBLER

W. BOWDISH

ANOTHER DISASSEMBLER FOR THE 1802

A long time ago (about 1978) I needed a small disassembler for a monitor which I was writing. The requirements where as follows:

- must be small, about 2 pages, since the monitor was only 2k bytes
- must print the output on a terminal
- must display the instruction address, mnemonic and operand fields
- should be a SCRT callable subroutine
- must output a specified number of instructions

This article describes a slightly modified version of that original disassembler. Since the original version was written, the routine has been modified and used in several applications (see T. Hills window program for a distant relative of this version).

OPERATION

The routine is a SCRT callable subroutine. On entry it expects some data in registers. R8 is assumed to contain a count of the number of instructions (not bytes) to be disassembled. R9 contains the start address of the instructions to be processed. In addition to these 2 registers, R7, RB and RC are used.

The method of disassembling instructions is fairly straight forward. For each instruction to be disassembled, the high nibble is used to index into a table (HIGTAB) which contains the address (low byte only) of a routine which will process that instruction type. Usually the instruction processing is straightforward, but much of the code is used to test for and handle the special cases.

The last two pages of the listing are tables which contain the ASCII mnemonics for the instructions. These tables contain either 3-byte or 4-byte entries. Note that the last byte of each entry has the high bit set. The routine 'STBXFR' copies these mnemonics from the table to the output device until a character with bit 7 set is encountered.

If you ever have a need for a small disassembler then this little routine may be of some use.

```
      =  01 00                  .ORG    #0100
                          ;
                          ;
                          ;                      D I S A S M
                          ;
                          ;                CDP1802 MINI-DISASSEMBLER
                          ;
                          ;     THIS SUBROUTINE DISASSEMBLES A SPECIFIED NUMBER OF
                          ;     INSTRUCTIONS AND PRINTS THEM ON THE TERMINAL.  THE
                          ;     CALLING SEQUENCE IS AS FOLLOWS:
                          ;
                          ;             ...         ; R8 CONTAINS A COUNT OF THE NUMBER
                          ;                         ; OF INSTRUCTIONS TO PROCESS
                          ;             ...         ; R9 CONTAINS THE STARTING ADDRESS
                          ;                         ; OF THE CODE TO BE DISASSEMBLED
                          ;        +CALL   DISASM
                          ;
                          ;
                          ;           E X T E R N A L   R O U T I N E S
                          ;
                          ;        TTYOUT - ROUTINE TO OUTPUT A CHARACTER TO THE TERMINAL.
                          ;                 THE CHARACTER IS PASSED IN THE D-REGISTER
                          ;
                          ;
                          ;             R E G I S T E R   U S E A G E
                          ;
                          ;        R0-R6   STANDARD
                          ;         R7     SPACING COUNT
                          ;         R8     NUMBER OF INSTRUCTIONS TO PROCESS
                          ;         R9     ADDRESS OF INSTRUCTIONS
                          ;         RA     ** NOT USED **
                          ;         RB     POINTER TO DATA TABLE PAGE
                          ;         RC.HI  HIGH NIBBLE OF INST. BEING PROCESSED
                          ;         RC.LO  LOW NIBBLE OF INST. BEING PROCESSED
                          ;         RD     ** NOT USED **
                          ;         RE     ** NOT USED **
                          ;         RF.HI  USED BY SCRT ROUTINES TO PASS D-RE.
                          ;         RF.LO  ** NOT USED **
                          ;
                                .SLW
```

```
                        ;           DISASSEMBLER ENTRY POINT
                        ;
                        ;     LOOP BACK HERE FOR EACH INSTRUCTION
                        ;
                        ;
0100                    DISASM:
0100  93                        GHI     R3      ;\
0101  FC 01                     ADI     1       ; POINT RB.HI TO THE DATA TABLE PAGE
0103  BB                        PHI     RB      ;/
0104  D4 01 25                  +CALL   CRLF    ; OUTPUT <CR><LF>
0107  D4 01 2E                  +CALL   PRTADR  ; OUTPUT INSTRUCTION ADDRESS
                        ;
                        ;     INSTRUCTION DECODER
                        ;
010A                    DECODE:
010A  F8 06                     LDI     6       ; SET UP THE SPACING COUNTER
010C  A7                        PLO     R7      ;
010D  09                        LDN     R9      ;\
010E  FA 0F                     ANI     #0F     ; PUT LOW NIBBLE OF INSTRUCTION IN RC.LO
0110  AC                        PLO     RC      ;/
0111  49                        LDA     R9
0112  32 8B                     BZ      IDLE    ;IDLE INSTRUCTION?
0114  F6                        SHR             ;\
0115  F6                        SHR             ; \
0116  F6                        SHR             ;  PUT HIGH NIBBLE OF INSTRUCTION IN RC.HI
0117  F6                        SHR             ; /
0118  BC                        PHI     RC      ;/
0119  FC 08                     ADI     HIGTAB  ;\
011B  AB                        PLO     RB      ; INDEX INTO HIGH NIBBLE BRANCH TABLE
011C  0B                        LDN     RB      ; AND BRANCH TO INSTRUCTION HANDLER
011D  A3                        PLO     R3      ;/
                        ;
                        ;    END OF INSTRUCTION PROCESSING
                        ;
011E                    ENDINS:
011E  28                        DEC     R8      ; DECREMENT THE INSTRUCTION COUNT
011F  88                        GLO     R8      ;\
0120  3A 00                     BNZ     DISASM  ; IF ALL INSTRUCTIONS HAVE NOT BEEN
0122  98                        GHI     R8      ; PROCESSED THEN LOOP FOR THE NEXT
0123  3A 00                     BNZ     DISASM  ;/
                        ;
                        ;     OUTPUT CARRIAGE ROUTINE, LINE FEED TO TERMINAL
                        ;
                        ;     NOTE: IF YOUR TERMINAL AUTOMATICALLY OUTPUTS A LINE
                        ;           FEED AFTER A CARRIAGE RETURN, THEN CHANGE THE
                        ;           BYTE AT LABEL $$$.1 TO A #D5.
                        ;
0125                    CRLF:
0125  F8 0D                     LDI     #0D
0127  D4 00 25                  +CALL   TTYOUT
012A  F8 0A             $$$.1:  LDI     #0A     ; *** SEE NOTE ABOVE ***
012C  30 38                     BR      CHROUT
                                .SLW
```

```
                              ;      PRINT THE INSTRUCTION ADDRESS
                              ;
012E                          PRTADR:
012E    99                            GHI      R9          ; OUTPUT HIGH BYTE OF ADDRESS
012F    D4 01 EA                      +CALL    HEX020
0132    89                            GLO      R9          ; OUTPUT LOW BYTE OF ADDRESS
0133    D4 01 EA                      +CALL    HEX020
0136    F8 20                         LDI      #20         ; OUTPUT A SPACE TO SEPERATE
                                                           ; THE ADDRESS AND MNEMONIC
                              ;
                              ;      CHARACTER OUTPUT ROUTINE - CALL FROM VARIOUS PLACES
                              ;
0138                          CHROUT:
0138    D4 00 25                      +CALL    TTYOUT
013B    D5                            +RETRN
                              ;
                              ;      DECODED INSTRUCTION OUTPUT ROUTINES
                              ;
                              ;      REGISTER TYPE INSTRUCTIONS ( XXX   RN )
                              ;
013C                          REG:
013C    9C                            GHI      RC                  ; GET HIGH NIBBLE
013D    D4 01 C6                      +CALL    UTLSUB              ; GOTO DISPATCHER
0140    C9                            .BYTE    INDEX3,REGTAB       ;  - INDEX INTO TABLE
0141    18
0142    F8                            .BYTE    REGSYM              ;  - OUTPUT REG. NUMBER
0143    30 1E                         BR       ENDINS              ; LOOP FOR NEXT INSTRUCTION
                              ;
                              ;      LONG BRANCH INSTRUCTIONS
                              ;
0145                          LBRS:
0145    F8 4C                         LDI      'L                  ;  NO - PRINT AN 'L'
0147    D4 00 25                      +CALL    TTYOUT
014A    27                            DEC      R7                  ; DECREMENT SPACER COUNT
014B    8C                            GLO      RC                  ; GET LOW NIBBLE AND
014C    FA 04                         ANI      #04                 ;  CHECK FOR LONG SKIPS
014E    3A 65                         BNZ      LSKP                ; SKIP INSTRUCTION?
                              ;
                              ;      SHORT BRANCH INSTRUCTIONS
                              ;
0150                          SBRS:
0150    8C                            GLO      RC                  ;\
0151    FB 08                         XRI      8                   ; CHECK FOR SKP INSTRUCTIONS
0153    32 94                         BZ       SKIP                ;/
0155    D4 01 C5                      +CALL    UTLGLC              ; GO TO DISPACTHER
0158    C9                            .BYTE    INDEX3,SBRTAB       ;  - INDEX INTO TABLETABLE
0159    48
015A    E4                            .BYTE    HEXBYT              ;  - OUTPUT OPERAND
015B    9C                            GHI      RC
015C    FA 0C                         ANI      #0C                 ; CHECK IF LONG BRANCH
015E    32 1E                         BZ       ENDINS              ;  NO  - THEN FINISHED
0160    D4 01 E9                      +CALL    HEX010              ;  YES - OUTPUT REST OF ADDRESS
0163    30 1E                         BR       ENDINS
                                      .SLW
```

```
                             ;
                             ;     LONG SKIP INSTRUCTION
                             ;
0165   8C                    LSKP:    GLO      RC              ; GET LOW NIBBLE
0166   FF 04                          SMI      4               ; SET UP OFFSET
0168   AC                             PLO      RC
0169   FF 04                          SMI      4               ; FIRST FOUR LSKP'S?
016B   33 6E                          BPZ      LSKP1           ;  NO  - OFFSET OKAY
016D   8C                             GLO      RC              ;  YES - RESTORE OFFSET
016E   D4 01 C6              LSKP1:   +CALL    UTLSUB          ; CALL DISPATCHER
0171   C9                             .BYTE    INDEX3,LBRTAB   ;  - INDEX INTO TABLE AND OUTPUT
0172   78
0173   C0                             .BYTE    OUTXIT          ; END
                             ;
                             ;     #6X INSTRUCTIONS
                             ;
0174   8C                    IOS:     GLO      RC              ; GET LOW NIBBLE
0175   32 91                          BZ       IRX             ; IS IT IRX?
0177   FF 08                          SMI      8
0179   32 8E                          BZ       ILLEGL          ; UNUSED         OPCODE?
017B   3B 81                          BNF      INP
017D   AC                             PLO      RC
017E   F8 2D                          LDI      SPEINP          ;
0180   C8                             LSKP                     ;
0181   F8 2A                 INP:     LDI      SPEOUT          ;
0183   AB                             PLO      RB
0184   D4 01 C6                       +CALL    UTLSUB          ;CALL DISPATCHER
0187   D0                             .BYTE    STBXFR          ; - OUTPUT NMEUMONIC ROUTINE
0188   FD                             .BYTE    OUTDIG
0189   30 1E                          BR       ENDINS
                                      .SLW
```

```
                         ;
                         ;    IDLE INSTRUCTION
                         ;
018B   F8 21             IDLE:   LDI     SPEIDL
018D   C8                        LSKP
                         ;
                         ;    ILLEGAL OPCODE
                         ;
018E   F8 45             ILLEGL: LDI     NOTUSD
0190   C8                        LSKP
                         ;
                         ;    IRX INSTRUCTION
                         ;
0191   F8 3C             IRX:    LDI     SPEIRX
0193   C8                        LSKP
                         ;
                         ;    SHORT SKIP INSTRUCTION
                         ;
0194   F8 60             SKIP:   LDI     SPESKP
0196   AB                        PLO     RB
0197   D4 01 C6                  +CALL   UTLSUB          ; CALL DISPATCHER
019A   D0                        .BYTE   STBXFR          ;  - OUTPUT NMEUMONIC
019B   C0                        .BYTE   OUTXIT          ;  - END
                         ;
                         ;    #7X INSTRUCTION
                         ;
019C   8C                MIS1:   GLO     RC              ; GET LOW NIBBLE
019D   FE                        SHL                     ;  X4
019E   FE                        SHL
019F   D4 01 C6                  +CALL   UTLSUB          ; CALL DISPATCHER
01A2   CD                        .BYTE   INDEX4,MS1TAB   ;  - INDEX INTO TABLE AND OUTPUT
01A3   90
01A4   C3                        .BYTE   UTLXIT          ;
01A5   FF 0C                     SMI     #0C             ; NEED AN OPERAND?
01A7   3B 1E                     BM      ENDINS          ;  NO - THEN BRANCH
01A9   FF 02                     SMI     2
01AB   30 B9                     BR      MS1RTN          ; MAYBE...
                         ;
                         ;    #FX INST
                         ;
01AD                     MIS2:
01AD   D4 01 C5                  +CALL   UTLGLC          ; CALL DISPATCHER
01B0   C9                        .BYTE   INDEX3,MS2TAB   ;  - INDEX INTO TABLE AND OUTPUT
01B1   D0
01B2   C3                        .BYTE   UTLXIT          ;  - END
01B3   FF 08                     SMI     8               ; NEED AN OPERAND?
01B5   3B 1E                     BM      ENDINS          ;  NO  -
01B7   FF 06                     SMI     6
01B9   32 1E             MS1RTN: BZ      ENDINS          ;  NO -
01BB   D4 01 E4                  +CALL   HEXBYT          ;  YES - OUTPUT OPERAND
01BE   30 1E                     BR      ENDINS
                                 .SLW
```

```
                        ;
                        ;       DISSASEMBLER UTILITIES
                        ;
01C0                    OUTXIT:                        ; EXIT FROM UTILITIES TO END PROCESSING
01C0    F8 1E                   LDI     ENDINS
01C2    A6                      PLO     R6
01C3    8C              UTLXIT: GLO     RC             ; NORMAL EXIT FROM UTILITIES
01C4    D5                      +RETRN
                        ;
                        ;       ENTRY POINTS TO UTILITY ROUTINES
                        ;
                        ;          UTLGLC - ENTER AND PUT RC.LO IN D-REG
                        ;          UTLSUB - NORMAL ENTRY POINT
                        ;
01C5                    UTLGLC:
01C5    8C                      GLO     RC
01C6                    UTLSUB:
01C6    52                      STR     R2             ; SAVE CONTENTS OF D-REG
01C7    46                      LDA     R6             ; GET ROUTINE ADDRESS
01C8    A3                      PLO     R3             ;  AND BRANCH TO THE ROUTINE
                        ;
                        ;       CALC. OFFSET INTO 3-BYTE TABLES
                        ;
01C9    F0              INDEX3: LDX                    ;CALCULATE AND LOAD TABLE ADDRESS
01CA    FE                      SHL
01CB    F4                      ADD
01CC    52                      STR     R2
                        ;
                        ;       INDEX INTO APPROPRIATE TABLE
                        ;
01CD    46              INDEX4: LDA     R6             ;\
01CE    F4                      ADD                    ; POINT RB TO START OF TABLE ENTRY
01CF    AB                      PLO     RB             ;/
                        ;
                        ;       OUTPUT INSTRUCTION MNEMONIC
                        ;
01D0    0B              STBXFR: LDN     RB             ; OUTPUT DATA
01D1    D4 00 25                +CALL   TTYOUT
01D4    27                      DEC     R7             ; DEC. SPACER COUNT
01D5    4B                      LDA     RB             ;\
01D6    FE                      SHL                    ; CHECK FOR LAST CHAR IN ENTRY
01D7    3B D0                   BNF     STBXFR         ;/
01D9                    STBX$1:
01D9    F8 20                   LDI     #20            ;\
01DB    D4 00 25                +CALL   TTYOUT         ; \
01DE    27                      DEC     R7             ;  SPACE OVER TO OPERAND FIELD
01DF    87                      GLO     R7             ; /
01E0    3A D9                   BNZ     STBX$1         ;/
01E2    30 C6                   BR      UTLSUB
                                .SLW
```

```
                        ;
                        ;       OUTPUT HEX BYTE PRECEEDED BY '#'
                        ;
01E4                    HEXBYT:
01E4    F8 23                   LDI     '#
01E6    D4 00 25                +CALL   TTYOUT
                        ;
                        ;       OUTPUT HEX BYTE POINTED TO BY R9
                        ;
01E9                    HEX010:
01E9    49                      LDA     R9
                        ;
                        ;       OUTPUT CONTENTS OF D-REG
                        ;
01EA                    HEX020:
01EA    73                      STXD
01EB    F6                      SHR
01EC    F6                      SHR
01ED    F6                      SHR
01EE    F6                      SHR
01EF    D4 01 FE                +CALL   HEXASC
01F2    60                      IRX
01F3    F0                      LDX
01F4    FA 0F                   ANI     #0F
01F6    30 FE                   BR      HEXASC
                        ;
                        ;       OUTPUT 'R' FOLLOWED BY A REGISTER NUMBER
                        ;
01F8    F8 52           REGSYM: LDI     'R                      ; OUTPUT AN 'R'
01FA    D4 00 25                +CALL   TTYOUT
01FD    8C              OUTDIG: GLO     RC
                        ;
                        ;       HEX TO ASCII OUTPUT
                        ;
01FE                    HEXASC:
01FE    FF 0A                   SMI     10
0200    C7                      LSNF
0201    FC 07                   ADI     7
0203    FC 3A                   ADI     #3A
0205    C0 01 38                LBR     CHROUT
                                .SLW
```

```
                        ;       TABLE OF ADDRESSE
                        ;
0208    3C              HIGTAB: .BYTE    REG
0209    3C                      .BYTE    REG
020A    3C                      .BYTE    REG
020B    50                      .BYTE    SBRS
020C    3C                      .BYTE    REG
020D    3C                      .BYTE    REG
020E    74                      .BYTE    IOS
020F    9C                      .BYTE    MIS1
0210    3C                      .BYTE    REG
0211    3C                      .BYTE    REG
0212    3C                      .BYTE    REG
0213    3C                      .BYTE    REG
0214    45                      .BYTE    LBRS
0215    3C                      .BYTE    REG
0216    3C                      .BYTE    REG
0217    AD                      .BYTE    MIS2
                        ;
                        ;       MOSTLY REGISTER T
                        ;
0218    4C 44           REGTAB: .ASCII   \LD\
021A    CE                      .BYTE    'N!#80
021B    49 4E                   .ASCII   \IN\
021D    C3                      .BYTE    'C!#80
021E    44 45                   .ASCII   \DE\
0220    C3                      .BYTE    'C!#80
0221    49 44           SPEIDL: .ASCII   \ID\
0223    CC                      .BYTE    'L!#80
0224    4C 44                   .ASCII   \LD\
0226    C1                      .BYTE    'A!#80
0227    53 54                   .ASCII   \ST\
0229    D2                      .BYTE    'R!#80
022A    4F 55           SPEOUT: .ASCII   \OU\
022C    D4                      .BYTE    'T!#80
022D    49 4E           SPEINP: .ASCII   \IN\
022F    D0                      .BYTE    'P!#80
0230    47 4C                   .ASCII   \GL\
0232    CF                      .BYTE    'O!#80
0233    47 48                   .ASCII   \GH\
0235    C9                      .BYTE    'I!#80
0236    50 4C                   .ASCII   \PL\
0238    CF                      .BYTE    'O!#80
0239    50 48                   .ASCII   \PH\
023B    C9                      .BYTE    'I!#80
023C    49 52           SPEIRX: .ASCII   \IR\
023E    D8                      .BYTE    'X!#80
023F    53 45                   .ASCII   \SE\
0241    D0                      .BYTE    'P!#80
0242    53 45                   .ASCII   \SE\
0244    D8                      .BYTE    'X!#80
0245    2A 49           NOTUSD: .ASCII   \*I\
0247    CC                      .BYTE    'L!#80
                                .SLW
```



```
                        ;       BRANCH INSTRUCTIONS ( AN
                        ;
0248    42 52           SBRTAB: .ASCII   \BR\
024A    A0                      .BYTE    ' !#80
024B    42 51                   .ASCII   \BQ\
024D    A0                      .BYTE    ' !#80
024E    42 5A                   .ASCII   \BZ\
0250    A0                      .BYTE    ' !#80
0251    42 44                   .ASCII   \BD\
0253    C6                      .BYTE    'F!#80
0254    42 31                   .ASCII   \B1\
0256    A0                      .BYTE    ' !#80
0257    42 32                   .ASCII   \B2\
0259    A0                      .BYTE    ' !#80
025A    42 33                   .ASCII   \B3\
025C    A0                      .BYTE    ' !#80
025D    42 34                   .ASCII   \B4\
025F    A0                      .BYTE    ' !#80
0260    53 4B           SPESKP: .ASCII   \SK\
0262    D0                      .BYTE    'P!#80
0263    42 4E                   .ASCII   \BN\
0265    D1                      .BYTE    'Q!#80
0266    42 4E                   .ASCII   \BN\
0268    DA                      .BYTE    'Z!#80
0269    42 4E                   .ASCII   \BN\
026B    C6                      .BYTE    'F!#80
026C    42 4E                   .ASCII   \BN\
026E    B1                      .BYTE    '1!#80
026F    42 4E                   .ASCII   \BN\
0271    B2                      .BYTE    '2!#80
0272    42 4E                   .ASCII   \BN\
0274    B3                      .BYTE    '3!#80
0275    42 4E                   .ASCII   \BN\
0277    B4                      .BYTE    '4!#80
                        ;
                        ;       LONG SKIP INSTRUCTIONS (
                        ;
0278    4E 4F           LBRTAB: .ASCII   \NO\
027A    D0                      .BYTE    'P!#80
027B    53 4E                   .ASCII   \SN\
027D    D1                      .BYTE    'Q!#80
027E    53 4E                   .ASCII   \SN\
0280    DA                      .BYTE    'Z!#80
0281    53 4E                   .ASCII   \SN\
0283    C6                      .BYTE    'F!#80
0284    53 49                   .ASCII   \SI\
0286    C5                      .BYTE    'E!#80
0287    53 51                   .ASCII   \SQ\
0289    A0                      .BYTE    ' !#80
028A    53 5A                   .ASCII   \SZ\
028C    A0                      .BYTE    ' !#80
028D    53 44                   .ASCII   \SD\
028F    C6                      .BYTE    'F!#80
                                .SLW
```

```
                          ;
                          ;       #7X INSTRUCTIONS
                          ;
0290  52 45 54    MS1TAB: .ASCII    \RET\
0293  A0                  .BYTE     ' !#80
0294  44 49 53            .ASCII    \DIS\
0297  A0                  .BYTE     ' !#80
0298  4C 44 58            .ASCII    \LDX\
029B  C1                  .BYTE     'A!#80
029C  53 54 58            .ASCII    \STX\
029F  C4                  .BYTE     'D!#80
02A0  41 44 43            .ASCII    \ADC\
02A3  A0                  .BYTE     ' !#80
02A4  53 44 42            .ASCII    \SDB\
02A7  A0                  .BYTE     ' !#80
02A8  53 48 52            .ASCII    \SHR\
02AB  C3                  .BYTE     'C!#80
02AC  53 4D 42            .ASCII    \SMB\
02AF  A0                  .BYTE     ' !#80
02B0  53 41 56            .ASCII    \SAV\
02B3  A0                  .BYTE     ' !#80
02B4  4D 41 52            .ASCII    \MAR\
02B7  CB                  .BYTE     'K!#80
02B8  52 45 51            .ASCII    \REQ\
02BB  A0                  .BYTE     ' !#80
02BC  53 45 51            .ASCII    \SEQ\
02BF  A0                  .BYTE     ' !#80
02C0  41 44 43            .ASCII    \ADC\
02C3  C9                  .BYTE     'I!#80
02C4  53 44 42            .ASCII    \SDB\
02C7  C9                  .BYTE     'I!#80
02C8  53 48 4C            .ASCII    \SHL\
02CB  C3                  .BYTE     'C!#80
02CC  53 4D 42            .ASCII    \SMB\
02CF  C9                  .BYTE     'I!#80
                          .SLW
```



```
                       ;
                       ;       #FX INSTRUCTIONS
                       ;
02D0  4C 44    MS2TAB: .ASCII    \LD\
02D2  D8               .BYTE     'X!#80
02D3  4F 52            .ASCII    \OR\
02D5  A0               .BYTE     ' !#80
02D6  41 4E            .ASCII    \AN\
02D8  C4               .BYTE     'D!#80
02D9  58 4F            .ASCII    \XO\
02DB  D2               .BYTE     'R!#80
02DC  41 44            .ASCII    \AD\
02DE  C4               .BYTE     'D!#80
02DF  53 44            .ASCII    \SD\
02E1  A0               .BYTE     ' !#80
02E2  53 48            .ASCII    \SH\
02E4  D2               .BYTE     'R!#80
02E5  53 4D            .ASCII    \SM\
02E7  A0               .BYTE     ' !#80
02E8  4C 44            .ASCII    \LD\
02EA  C9               .BYTE     'I!#80
02EB  4F 52            .ASCII    \OR\
02ED  C9               .BYTE     'I!#80
02EE  41 4E            .ASCII    \AN\
02F0  C9               .BYTE     'I!#80
02F1  58 52            .ASCII    \XR\
02F3  C9               .BYTE     'I!#80
02F4  41 44            .ASCII    \AD\
02F6  C9               .BYTE     'I!#80
02F7  53 44            .ASCII    \SD\
02F9  C9               .BYTE     'I!#80
02FA  53 48            .ASCII    \SH\
02FC  CC               .BYTE     'L!#80
02FD  53 4D            .ASCII    \SM\
02FF  C9               .BYTE     'I!#80
                       .END
```

USING THE VDU BOARD FOR RAM ONLY

-by G. F. Feaver, Burlington, Ontario

A check of the schematic for the VDU board will reveal that if the 6847 socket is not populated, all "B" inputs to the 4-4019 IC's will be floating. This is not a desireable condition.

This could not only cause an erroneous output but could also destroy the chip. RCA in "COS/MOS MEMORIES", (page 13), states that a floating input on some IC's such as the 4049 and 4050 can cause the maximum power of 200 mw to be exceeded and may result in damage to the device. Fairchild in their "CMOS DATA BOOK", (page 5-9), states that "all unused INPUTS must be tied to VCC or Ground less they generate a local "MAYBE". The bad TTL habit of leaving unused inputs open is definitely out."

Motorola in McMOS HANDBOOK, (Page 6-10) states that "...by considering the numerical values of the equivalent capacitors and equivalent resistors determined by the PN junctions (of the inputs), if can be seen that the input potential of non-connected inputs is not well defined. This fact can bring the transistors into operation and generate false output operation. Consequently, all unused inputs shuld be tied either to ground or to VDD depending on the required logic function.
'Unused input of NAND gates should be tied to VDD
'Unused inputs of NOR gates should be tied to VSS (Ground)"

Thus unused inputs of AND gates should be tied to VDD and unused inputs of OR gates should be tied to VSS (ground).

The same comment applies to unused logic gates in a package which can generate perturbations in a system through the power supply line.

It is thus recommended that all pins 1, 3, 5, 7, of 4019 IC's #2, #3 and #-#4, be connected to ground through a resistor (10k to 100k ohms) and pin 7 of 4019, #5 be connected likewise. Pins 1, 2 and 3 of 4019 chip #5 are connected to inverter #9 and pin 5 is connected to +5 and should be satisfactory.

On 4049, #9 connect pin 11 to pin 10 of the same chip.

A Minimum Count 2114 Memory System Using the VDU Board

- Fred Feaver, 103 Townsend, Burlington, Ontario

For those who purchased a VDU board and then temporarily shelved it due perhaps to the high cost of the MC6847 colour chip or for other reasons, but would still like to have an inexpensive 16K of 2114 memory (remember the 2114L draws 30% less power than the 2114 chip), the following article should be of interest. This is a minimum parts system.

My micro is a Tek1802 but the information should be usable with any system.

I had soldered in all my sockets before deciding that I would not use the video portion of the board. I noted on studying the schematic that the 4019 quad double pole selector chips would have one input to each "B" AND gate (in the chip) left floating if the 6847 chip was not inserted into its socket.

According to manufacturers' recommendations, this must not be done: all inputs must be connected into circuit or else connected to VCC or VSS as appropriate. This information meant that suitable resistors would be required for termination of the floating inputs. It was also noted that one inverter - 11/12 - of the 4049 was left floating. the necessary resistors were connected in place.

With the realization that a number of unnecessary chips would be drawing power if the board was used for memory, I decided to strip the unnecessary chips and resistors and go for a minimum count system as listed below, the modified schematic is included. Note that the pin markings on the 2114 section for D0 thru D7 do not agree with the original schematic. The original schematic should have read like the modified one. (A pin to pin check was made.)

Refer to "Original" parts layout and schematic supplied with PC board.

1. Remove sockets for IC's #1 to 5, #7, #9 and #11 to 14, #11, #12 (this is really not necessary but it makes for a cleaner, easier soldering operation).

2. Cut trace close to pin 23 of IC#6.
   Cut traces close to pins 3, 4, 5, 6, 8, 9, 10, 11, 12, 13 of IC#10.

3. Connect the following jumpers:
   IC#10 pin 3 to 5 to 6
   IC#10 pin 10 to 12 to 13
   IC#6 pin 23 to IC#10 pins 4, 8 and 9

4. Connect a jumper from IC#9 pin 15 to IC#9 pin 3
   (Note IC#9-404-9 Chip must not be inserted in its socket)
   (This jumper connects Memory chip #10 to buss H$(\overline{MRW})$)

5. Connect the following jumpers:

IC's #2, #3 and 4 (for each chip)	Jumper pin 2 to 12
	4 to 11
	6 to 10
	15 to 13
IC#5	Jumper pin 4 to 11
	6 to 10
IC #11 and IC #12 (for both chips)	Jumper pin 1 to 2
	3 to 4
	8 to 9
	10 to 11

NOTE: **The above conversion leaves active connections to the unused sockets so if these sockets are used for other functions at a later date, then more traces must be cut. These can be seen from the "original" schematic.**

On completion of the above changes and before inserting IC chips, check for shorts between busses. (Edge connector fingers 1 and 22). If resistance is less than about 1 megohm, find and correct the fault.

Make a careful visual check to determine that all changes have been made as specified.

If the above checks are satisfactory install the support chips as follows:

Install a 4001 quad 2 input NOR gate in socket for IC#10
Install a 4508 quad in socket for IC#8.
Install a 4515 in socket for IC#6.

(NOTE: **IC#10 only requires an OR gate but spare OR gates are not as useful as NOR gates, hence the substitution.**)

Plug the board into Tek 1802 Motherboard and POWER UP. If power supply light does not go out and you do not feel any unduly warm chips, your currents are probably OK. (Remember the Tek 1802 power supply is limited against overcurrents up to about 1 ampere).

POWER DOWN and remove the VDU board from the Motherboard. Install the 2114 Memory chips starting at position 0 and filling locations sequentially, being careful about location of pin 1.

Plug the VDU board into the Motherboard. Remove the 3/4 K MB1 board and the 2-2101 chips from the micro board.

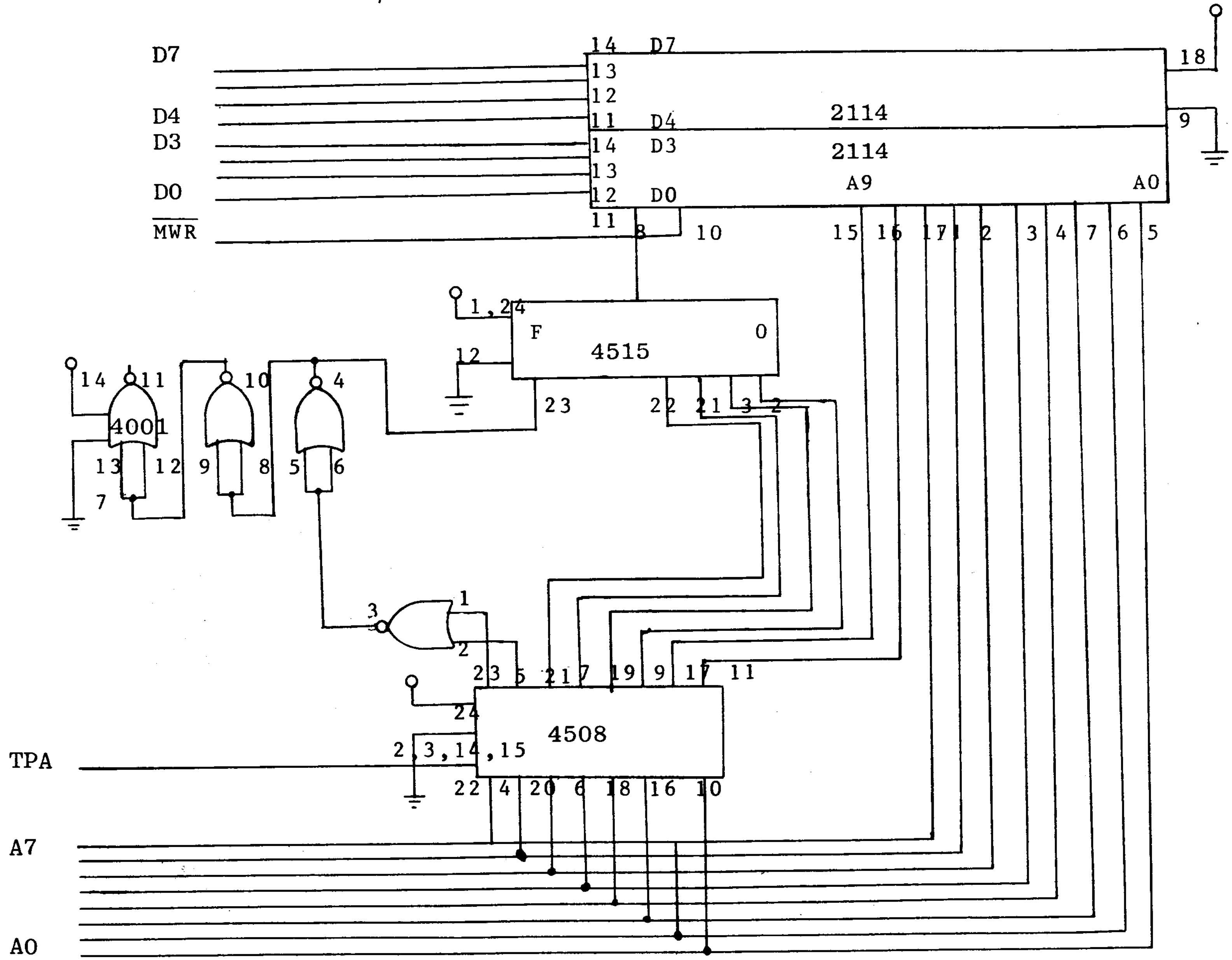
POWER UP

Run the memory check given in the VDU documentation or from IPSO FACTO #4, page 20 or from DEFACTO, page I-65.

This Mod will give up to 16K of 2114 RAM memory from 0000 to 3FFF. No buffers have been used and no trouble has been experienced.

Use of the space NOR gates as inverters to A14-A15 will relocate the 16 block to any connection desired.

ACE VDU/MEMORY BOARD



Netronics Text Editor Improvements

-by Al Irwin, 1312 W.Hill St., Champagne, Ill. 68120

Mr. Eric Tyson had a patch to a print routine in your July issue. I called him and we had a fine conversation. It turns out that what he did works, but he did not know how the link to and from his routine worked. He did not know what the GHI R0 (first byte of his listing) was for.

I have seen negative comments about Netronics because they do not offer help with that software. The reason probably is because they may not know it's workings either. I think (after conversing with RCA software people) that the editor was written at RCA. After I broke it down, it appears that it was originally written for disk as well as tape. RCA probably made patches in the source and assembled it for Netronics.

The reason this appears so, is that the space from 0E52 to 0F51 is a 256 byte I/O buffer. Eric is correct, from 0F52 to 0FFF is not used. The buffer is actually two 128 byte buffers, one for outgoing, (0ED2), one for incoming data, (0E52). Note that 128 is a nice round disk sector. When you get the editor from Netronics, you will find "George" in the buffer several times. Being that George is the name of "number one" at Netronics, I suspect that he tested the software by making a small file with his name in it a few times, then wrote it out to tape and read it back in. They then made tapes of that tested version so his name ends up in all copies sold by them. This is a deduction on my part and may not be 100% correct, but probably is not too far off.

I once heard that a Mr. Larry Sandlin is the author of the editor but I never researched it. I think the author deserves a big thanks as it IS a high level piece of software, even if it IS patterned after TECO, the editor that DEC used in the PDP-8 era. I detest the escape key being the command delimiter however, as most modern text editors use the return key. It is better with the CRT type terminal.

If you do some tape I/O, you will note the contents of the buffer has changed. If you load the editor, you can erase 0E52 to 0FFF and it will run just as well. You will also note that when reading or writing tape, that your display seems to be active in "bursts". Each burst is a 128 byte chunk of your file on the move. If you change your tape I/O in ROM to a disk I/O, the editor would not care. The final text buffer starts at 1020 and runs upward. Any I/O is copied from there to the 0E52/0F52 buffer or from that buffer into the final buffer area depending on data direction.

The space from 1000 to 1020 is a buffer log, where pertinent information about your file is kept. The location of start of file, end of file, current line start and end, start and end of "save", width of terminal in characters, (32, 64 or 80), location of current cursor, location of start and end of command buffer and other things are kept there. The location Eric chose is the curent cursor, so if

If you do a "P", you enter at 0A50, do a "done with chain, execute next bytes" which is GHI R0. Next you GHI R1, drop the high bit, and PHI R1. You then D407CD which is call chain, link to "load link register", (80), and label for link register is 035E which is back in chain at command level just the same as Eric did at 098A. This means that when you use the print command, you simply drop the top bit in R1 and return to the command level for the next command. If you do a "T", you run through it's string of one-byte links then end it with the same drop top bit in R1 and return to command level by falling through the "P" link string.

I have often wondered if the "drop high bit in R1" part of the print routine had to do with checking the high bit in a parallel output port which could have been used as a status bit to see if the device was on and ready to print, if not it would have ignored the command.

If this were true, the "T" command probably did not "fall through" the string of "P" links. Since there were a couple of bytes left unused after the change, it could indicate that there was a return to chain at the end of the "T" link string.

A patch that you may find useful, is one to eliminate the loss of your file in buffer after you tape it. I found that on occasion, a "drop out" in tape would cause an error. If after you tape the file, if you try to check it by reading it back, and get "tape error", you loose a lot of time trying to load it back and fix it. If you could tape it, and merge it back to the end of the file you just wrote out, the tape can be verified. If an error exits, just delete what you merged back and retape it again. This patch is for the "Y" command, the "Q" still blows away the file and "W" still removes that part which has been written out to tape.

The patch is:

Put a 30 AA at 0587 and 0588. At 05AA, put D4 07 CD 80 03 5E. What you are stepping on at 05AA is several bytes of another "island" of unused code left over from another "modification". There are several of these in the listing. While I am at it, the byte at 03EA should be 0D, not 8D. It was wrong on the Netronics tape. This is a CR in an ASCII string.

It is obvious that the Netronics editor was written with the hard copy terminal in mind, such as the Teletype Model #33 or equivalent type. I say this because it uses the escape key for the command delimiter and delete or rubout as the backspace. On a terminal which has hard copy, as you "rubout", you usually echo the deleted character to the terminal. You do not backspace the terminal, as any further input would then type over the text on the paper. This was fine for that mode of input.

you are at a given line in the file, your cursor will be at the start of the line so his print routine will start at that point in the file. If you do a "find" for a word within the line, the cursor will be at the end of the word when located. If Eric does a print after doing a find, he will start printing at current cursor, following the word. I did a similar thing to what Eric did to get a print routine on the editor, except that I made my call in the 0A50/0A5A part of the code. The two bytes at 0A5B and 0A5C can also be changed and the space used for your patch, as this is an "island". The two bytes are left over from their patch change when print was patched out. My print routine looks at the buff-log at 1016 to get the location to begin printing from, as that location holds the start of current line and is up-dated when you do a "01t", which is "show me the current line".

The editor makes use of what is called a "chain" and "link" system. When it does a task using chain, the call to chain is followed by one-byte links to most any subroutine in the editor. When the editor is at command level, it is in the chain routine. As it scans the command table, each command look-up is followed by a two-byte label to load into it's "link" register. Eric changed the two-byte label to be loaded into the link register for the "P" command, this was at 03A2, directing it to 0F52. His listing then appeared at that location.

This works very well, but when chain is linked to 0F52 the call to that subroutine is still under the control of chain and expects to find a string of one-byte links at that location. If you want executable code at that location instead of one-byte calls, you must exit chain. The author had need to do this and built in the process, which is the GHI R0, or 90. This is a one-byte call to the "done" routine which says we no longer want to be in chain, so exit chain and do an "execute immediate". So, the 90 at the start of Eric's print routine will cause the code following it to be run. His return is a call to 098A at 0F80 in his listing. If you look at 098A, you find D407CD, which is a call to chain followed by 80 which is a one-byte link to the "load link register" routine, the 035E following the 80 is the label for the link load which is a location back in the command level.

This means that when Eric exits his print routine, he goes back to chain at the command level and all is well. I commend Eric for his efforts, he did a good job, even if all of the patch was not fully understood.

A true fact about the editor is that there is no print routine in it, what it used to be was patched out. The "T" command link string is at 0A4A. It is a string of one-byte links for chain. It ends at 0A4F. The 14 and 16 appearing in the string are one-byte labels for buffer pointers. They point to 1014 and 1016 in the buff-log. At 0A4F, (last "T" link) there is a 7C, the next byte is the magic 90. This is at 0A50 and is the start of the "P" command link string. The "T" command continues through the "P" link string so the "P" string is used every time you use the "T" command.

If you have a CRT type terminal however, it is a nuisance because as you delete characters, your terminal cursor moves right with the echoed deleted characters and your editor buffer pointers move left in respect to your terminal display. This means that your terminal cursor is not really telling you where your are on the line.

A fix for this, is to patch the editor to recognize the backspace key, and also shut off the echoed characters that have been deleted. Your editor and terminal then remain together with their pointers.

A patch for this is:

At 029D-change 76 to 01, at 02A1-change 72 to 05, at 02DD-change E8 to F8, at 02DE-change F0 to 00 and at 02DF-change 3A to 30. This completes the backspace patch. For terminals without the backspace key, the "Control H" is used.

I also enjoy breaking down systems others have written, for the fun and challenge. It beats a crossword puzzle any day. I wrote a complete source for the Netronics Editor, and understand every byte in it. I being a professional in the field, understand the value of protecting copyrights and would not undermine Netronics by making it public. I can offer advice to anyone needing patches for that editor if they were to write to me, return postage included.

Netronics Compatible Tape Load Program

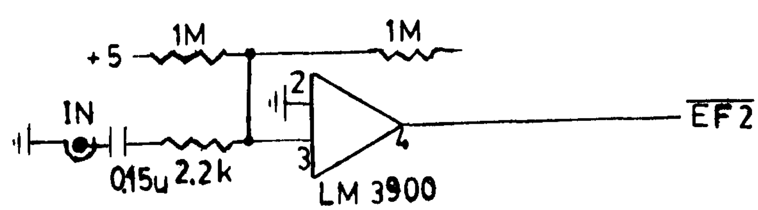
by M.E. Franklin, 690 Laurier Ave., Milton, Ontario, Canada. L9T 4R5

The program listed below will load any cassette SAVED using Q and the Netronics Cassette Software, including any software distributed by ACE.

- Load R8 with start address
- Load RA with dump length (plus 1 page)
- Load loader program into RAM not within dump area
- Jump to loader program via RO
- Q led will come on when loading is complete
- Hex leds will display addresses being loaded

```
0000   F800   LDI #00        0028   8D     GLO RD
0002   B8     PHI R8         0029   F6     SHR
0003   F800   LDI #00        002A   3B2F   BNF #2F
0005   A8     PLO R8         002C   7B     SEQ
0006   F817   LDI #17        002D   302C   BR  #2C
0008   BA     PHI RA         002F   9D     GHI RD
0009   F800   LDI #00        0030   58     STR R8
000B   AA     PLO RA         0031   E2     SEX R2
000C   90     GHI RO         0032   88     GLO R8
000D   B9     PHI R9         0033   22     DEC R2
000E   F83F   LDI #3F        0034   52     STR R2
0010   A9     PLO R9         0035   64     OUT R4
0011   F8F9   LDI #F9        0036   18     INC R8
0013   BD     PHI RD         0037   2A     DEC RA
0014   D9     SEP R9         0038   9A     GHI RA
0015   3B11   BNF #11        0039   3A1A   BNZ #1A
0017   9D     GHI RD         003B   302C   BR  #2C
0018   3A14   BNZ #14        003D   1D     INC RD
001A   D9     SEP R9         003E   D0     SEP RO
001B   331A   BDF #1A        003F   F80D   LDI #0D
001D   F801   LDI #01        0041   3541   B2  #41
001F   AD     PLO RD         0043   353D   B2  #3D
0020   BD     PHI RD         0045   FF01   SMI #01
0021   D9     SEP R9         0047   3343   BDF #43
0022   9D     GHI RD         0049   3D49   BN2 #49
0023   7E     SHLC           004B   303E   BR  #3E
0024   BD     PHI RD         004D   00     IDL
0025   3B21   BNF #21        004E   FFFF   SMI #FF
0027   D9     SEP R9
```

Netronics Compatible Cassette Hardware



CLUB COMMUNIQUE

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	______
I/O Adapter for Backplane, Ver. 1	______	20.00	______
64K Dynamic (4116) Board	______	50.00	______
EPROM (2716/32) Board	______	40.00	______
Kluge (wire wrap) Board	______	25.00	______
8" Disk Controller Board	______	40.00	______
Netronics - Ace Adapter Board	______	25.00	______
Netronics - Quest Adapter Board	______	20.00	______
DMA Adapter Board (ELF II)	______	3.00	______
VDU Board	______	40.00	______

Software

Fig FORTH - Netronics Cassette	______	$10.00	______

Back Issues

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

Membership

Current Year - Sept. '82 - Aug. '83
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.