Ipso Facto

ISSUE 39 February 84

INDEX PAGE

A PUBLICATION OF THE ASSOCIATION OF COMPUTER-CHIP EXPERIMENTERS

1983-1984 EXECUTIVE OF THE ASSOCIATION OF COMPUTER CHIP EXPERIMENTERS

President: John Norris 416-239-8567 **Vice-President:** Tony Hill 416-876-4231

Treasurer: Ken Bevis 416-277-2495 **Secretary:** Fred Feaver 416-637-2513

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

Newsletter:

Production Manager: Mike Franklin 416-878-0740

Editors: Fred Feaver
Tony Hill

Publication: Dennis Mildon
John Hanson

Product Mailing: Ed Leslie 416-528-3222 (Publication)

Fred Feaver 416-637-2513 (Boards)

Club Mailing Address:

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

ARTICLE SUBMISSIONS:

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

PUBLICATION POLICY:

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

MEMBERSHIP POLICY:

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

EDITOR'S CORNER

In last issue, I commented on the lack of article submission from you, our members. Well the responce has, again, been disappointing.

Yes, I did receive several articles, very good articles, as found in this issue. In addition, I received the article on CHIP 8 AE, which is not included since we have not yet verified the code. But, I did not receive enough articles to fill this issue, nor are their any in reserve for sthe next.

At the last Executive meeting, the Executive discussed moving on to as new micro chip, probably a 16 or 32 bit devise, and abandoning the 1802. This will mean discarding several hundreds of dollars worth of hardware and all our software, and investing potentially several thousand dollars in new hardware and software. Then comes the question of how many of our existing members would follow suit? I do not believe that many of you would.

Perhaps it is time to move the executive of ACE from the jolly band at Toronto to another city where a number of 1802 enthusiasts live. Then the Toronto mafia can move on th what ever micro they please, and the club will live on. Any volunteers?

Your comments will be welcome.

In the next issue,the major article will be on CHIP 8, and the numberous additional commands added by Larry Owen, the author of Alien.

FOR SALE: Complete Netronics ELF II system

Includes: ELF II motherboard with five 86-pin connectors
Giant I/O board
32K user RAM (on two 16K boards)
EPROM Programmer board
A.C.E. Netronics Adapter Board with six 44-pin connectors
A.C.E. Video Display Board with 8K RAM
Epson printer interface board
Enclosure for above
Netronics ASCII keyboard with cabinet
Power Supply (±8V DC, 6.3V AC)
Speaker/amplifier
Extensive software, literature, periodicals, documentation.

Everything except ASCII keyboard works. (I believe keyboard needs a new encoder chip.)
Worth over $1250.
Will sell separately.
$625 or best offer.

George Musser,
60 Broadway Rd.,
Warren, N.J., USA 07060
phone: 201-647-7939

Doug Moyer, 1856 Pacific Avenue, Winnipeg,Man. R2R 0G1

I have just finished reading the Editor's Corner in the December 1983 issue. I would likely buy an A/D and D/A board if they went into production. It would be a good idea to add optoisolated input and output ports. This would be useful for anybody that plans to use the A/D part as feedback for some sort of conputer control. The board, in either form, would be a nice addition to my robot system.

I have been working on a 300 Baud modem for the past few months. It is an originate/answer modem using the LM567 PLL for the receive section and the Exar 2207 function generator for the transmit section.

In the last several issues there has been a question as to whether the majority of the members want to develop their 1802 computer into a high level system. Since the main subject of all the issues of Ipso Facto deal with the improvment of a high level system, the lack of article submissions must prove that the majority don't want a high level system. If so, how can the number of articles submitted be increased?

Tutorial articles would help the low tech members. If members could be persuaded to write about their current project, future plans, problems encountered, etc. then other members might start working on the same idea or solve the problem. There must be some members out there that do the same thing that I do. They work on their pet project (mine is robotics) without telling anybody what they have accomplished. If these

people write articles then I'm sure other people will become interested in the same project.

Some ideas for projects that Ican think of are: robotics, weather station, and computerized house control. Other ideas have been discussed in some previous issues. The ideas mentioned can all be divided into smaller sections.

Robotics has some very challenging subsections that are just in the pioneer stages of development. Artificial Intelligence plays a very important role in robotics, but this is such a large area. It covers such things as pattern recognition, voice recognition, and process learning and modification to name just a few. Computer vision is also in the pioneer stages of development. These subsections would challenge the high tech members that have advanced design and programming skills. The low tech members could design power distribution systems, motor control circuits, the mobile platform, and programs that use the circuits developed.

There is a problem with the projects I have presented; like the high level system, the projects need expesive components. If I understood the criticism about the high level system correctly as, members don't want to make what they can buy, then the projects mentioned would soon have similar criticism. So what can be made with a computer that hasn't been done before? Somebody has even computerized a woodstove (Ciarcia, Steve Byte February, 1980)

1802 ROBOT PROJECT

For the past few years I have been experimenting with a homebuilt robot that is based on an 1802 computer. To this point in time I have finished the platform and a few sensors.

Through this series of articles I will explain the work that has been completed on the platform, the sensors, and the software that runs the system. I will also mention what I will be working on in the future. If anybody has interfaced their 1802 system to some real world devices they should describe the interface because it might give other people ideas for interfaces. Somebody might get stimulated to start working on a robot or some other interesting project of their own.

I would recommend that anybody who is planning to build their own robot, read at least one of the following books to give you a general outline of what is required for this project. By reading some of the books you also get the experience that the author has gained from the project without going through the same problems.

Books that cover the hardware aspect of this project very well are:

"Build Your Own Working Robot"
Author: David D. Heiserman
Book number: 841
Published by: TAB Book Club. . . . .
Although there is no computer in the project, it does have good ideas for sensors, including a circuit, that allows the robot to connect itself to a battery charger without human help.

"How to Build a Computer Controlled Robot"
Author: Tod Loofbourrow
Book number: 5681 - 8
Published by: Hayden Book Company, Inc. . . . .
This robot has a computer (a KIM) that has some sophisticated sensors such as an ultrasonic obstacle avoidance system and primitive voice recognition.

"How to Design & Build Your Own Custom Robot"
Author: David L. Heiserman
Book number: 1341
Published by: TAB Book Club. . . . .
This book breaks the robot down into small blocks and gives different circuits that could be used. For example one chapter entitled "Batteries and Power Supplies" discusses such topics as: advantages and disadvantages of different types of batteries (NiCad, Gell Cells, and lead acid cells.) estimation of battery capacity requirements power distribution sytems, plus others.

Good ideas for the software side of this project are available in the following books:

"How to Build Your Own Self-Programming Robot"
Author: David L. Heiserman
Book number: 1241
Published by: TAB Book Club. . . . .
This book develops an adaptive robot that can be "molded" by its environment.

"Robot Intelligence . . . With Experiments"
Author: David L. Heiserman
Book number: 1191
Published by: TAB Book Club

"Projects in Machine Intelligence For Your Home Computer"
Author: David L. Heiserman
Book number: 1391
Published by: TAB Book Club. . . . .
These two books cover the same ground as book number 241, but in a high level language. These books produce a series of robot simulations in Basic.

The following books cover some of the theory related to robots:

"Android Design"
Author: Martin Bradley Weinstein
Hayden book Company, Inc.
Number: 5192 - 1. . . . .
This book explains the theory of advanced sensors and motivation systems.

"Artificial Intelligence"
Author: Neil Graham
TAB Book Club
Number: 1070. . . . .
This book explains the theory of making computers seem to have intelligence.

The following books cover subjects that are related to robots:

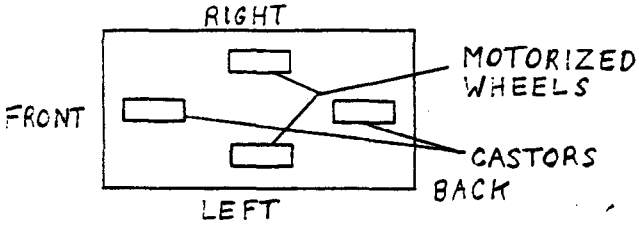
"Handbbok of Remote Control and Automation Techniques"
Author: John E. Cunningham
TAB Book Club
Number 1077

"Microcomputer Interfacing Handbook - A/D and D/A"
Author: Joseph J. Carr
TAB Book Club
Number: 1271

"Teaching Your Computer to Talk - A manual of command and response.
TAB Book Club
Number: 1330

"Verbal Control With Microcomputers"
Author: Mike Rigsby
TAB Book Club
Number: 1468.

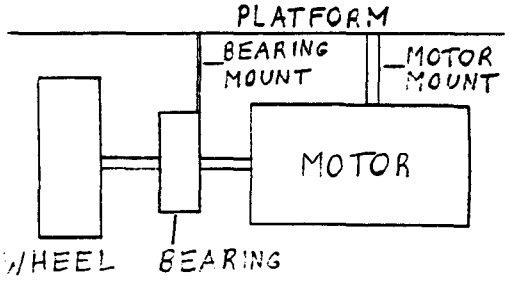
The mechanical platform will hold the batteries and the electronics of the robot. This will require the platform to be fairly sturdy and stable. Heavy objects, such as batteries, should be installed as close to the floor as possible to increase the stability of the platform. The most stable layout of the wheels is as follows:



The two motorized wheels are separate to allow the robot to turn in either direction and the two castors are for stability.

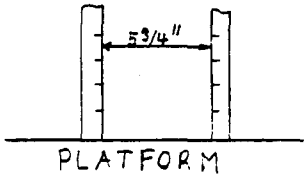
The motors needed for the robot should have high torque and low speed. I used Dodge power window motors. They have extremely high torque and turn slow enough to use six (6) inch diameter wheels.

The motors should be mounted so that most of the weight of the robot is resting on roller bearings to avoid etching a groove into the motor shaft.



For people that don't want to make their own, there is U.S. company that makes (see reference) robot platforms. Although the platform is good, I don't think that the price can be justified.

To mount the computer and electronic interfaces I made a wooden rack.



I happened to make the rack the same width as the ACE boards and I am now in the process of changing from an ELF II system to the ACE system.

The only thing that is needed to complete the platform is a portable power source. The only logical type of power source to use is batteries.

Nicads are plentiful but they have a few major drawbacks. The voltage across a nicad remains fairly constant until it is near complete discharge. At that point the voltage drops rapidly. This characteristic could strand the robot by surprise. The second drawback of nicads is that they can't be recharged at a high rate. This low rate means the robot will spend most of its time recharging its batteries. The third drawback is that the capacity of one battery is low. This can be overcome by paralleling more nicads but then a complicated recharging system is required to prevent the batteries from being overcharged or undercharged.

Using a lead-acid battery is a better choice for the power source. It has a larger charge capacity and can be recharged faster than nicads can. Lead-acid batteries should not be "fast charged" repeatedly because the life expectancy of the battery will be drastically reduced.

The major problem with lead-acid batteries is that when they are being charged or discharged rapidly, gases are given off which oxidize and dissolve PC board traces. To protect the PC boards from corrosion, encase the battery completely and have good ventilation away from the PC boards.

Another disadvantage is that lead-acid batteries require the addition of water at each recharge. They must also be kept upright to avoid spilling acid.

Maintenance free lead-acid batteries have solved the problems of corrosive gases being given off, the need to add water at each recharge, and spilling acid if the battery is knocked over. This kind of lead-acid battery would be a good alternative for the next battery to be explained.

The best choice for the power source are gell cells. This is a different type of lead-acid battery. It requires no maintenance, very little corrosive gases are given off, and nothing will spill out if the cell is knocked over. Gel cells also have a higher energy capacity per unit of weight than lead-acid batteries.

Addresses of companies mentioned:

Hayden Book Company, Inc.
Publisher
50 Essex St.
Rochelle Park, N.J.
07662

Hobby Robotics
(Platform Manufacturer)
5070 Buford Highway
Norcross, Ga.
30071

TAB Book Club
Publisher
Blue Ridge Summit, PA.
17214

Improvement to Dynamic RAM Board

Doug Moyer, 1856 Pacific Avenue Winnipeg, Man. R2R 0G1

I recently decided to upgrade from an Elf II system to an ACE system. When I bought the dynamic board I overlooked the fact that it required +12 volts. Since my system will be powered by a 12 volt battery. The memory would likely work incorrectly due to the noise generated by the high current motors. Thus the board couldn't be used. There is a very simple remedy to my problem, use 5 volt dynamic RAMs! The 4116 three supply RAM is interchangeable with the single 5 volt supply 2118 RAM. No PC board modifications are required!

SUPER WIPE OFF FOR CHIP VDU

BY - T HILL , 30-481 PITFIELD RD , MILTON , ONTARIO

A FEW MONTHS AGO , I REWROTE RCA'S CHIP-8 INTERPRETER TO RUN ON THE ACE VDU BOARD. THE PROGRAM WAS PUBLISHED IN IPSO FACTO ISSUE 35 (JUNE/83). ITS MAIN FEATURE WAS THE CHANCE TO USE A HIGHER RESOLUTION SCREEN THAT THAT OFFERED BY AN 1861.

MY FIRST PROJECT AFTER COMPLETING THE NEW CHIP-VDU WAS TO REWRITE ONE OF THE EXISTING GAMES. I CHOSE "WIPE OFF", FROM THE ORIGINAL RCA MANUAL. IT IS BASICALLY A PADDLE GAME, LIKE BREAKOUT, EXCEPT THE OBJECT IS TO KEEP A BALL IN PLAY UNTIL ALL THE TARGETS ON THE SCREEN HAVE BEEN HIT.

MODIFICATIONS TO THE ORIGINAL GAME INCLUDED A NEW SCREEN LAYOUT THAT INCLUDES BOUNDRY WALLS, RANDOM BALL BOUNCES FROM TARGETS AND THE ABILITY TO PUT 'ENGLISH' ON THE BALL. THE PADDLES GOES LEFT WITH KEY 4, RIGHT WITH KEY 6 AND THE NEXT BALL IS FIRED WITH KEY 0. F RESTARTS THE GAME AT THE END.

```
200  A3 20 62 10 61 20 D1 2F
208  71 08 31 60 12 06 72 10
210  32 30 12 04 A3 1E 62 0C
218  61 1F D1 21 71 01 31 61
220  12 1A 61 0D 62 1F 63 60
228  D2 11 D3 11 71 01 31 38
230  12 28 66 00 67 14 22 EE
238  22 F8 A3 1F 68 3C 69 38
240  D8 91 63 37 6D 03 8D 84
248  6F 00 EF 9E 12 4A A3 1E
250  DD 31 65 FF C4 01 34 01
258  64 FF A3 1F 6C 00 6A 04
260  EA 9E 12 68 38 20 6C FF
268  6A 06 EA 9E 12 72 38 5A
270  6C 01 D8 91 88 C4 88 C4
278  D8 91 4F 01 12 CA 4D 20
280  64 01 4D 5F 64 FF 43 0D
288  65 01 43 39 12 DE A3 1E
290  DD 31 8D 44 83 54 DD 31
298  3F 01 12 5A 43 38 12 CA
2A0  6A 02 F8 18 22 F8 76 01
2A8  22 F8 A3 1E DD 31 CA 03
2B0  7A FF 3A 02 84 A0 C5 01
2B8  35 01 65 FF 36 80 12 5A
2C0  6F 0F EF 9E 12 C2 00 E0
2C8  12 00 6A 04 FA 18 A3 1E
2D0  DD 31 73 FF DD 31 65 FF
2D8  3C 00 84 C0 12 5A A3 1E
2E0  DD 31 22 EE 77 FF 22 EE
2E8  37 00 12 42 12 C0 A3 30
2F0  6A 02 6B 02 F7 33 13 00
2F8  A3 30 6A 60 6B 02 F6 33
300  F2 65 F0 29 30 00 DA B5
308  7A 05 F1 29 40 00 31 00
310  DA B5 7A 05 F2 29 DA B5
318  00 EE 00 00 00 00 80 F8
320  44 00 00 00 44 00 00 00
328  44 00 00 00 44 00 00 00
```

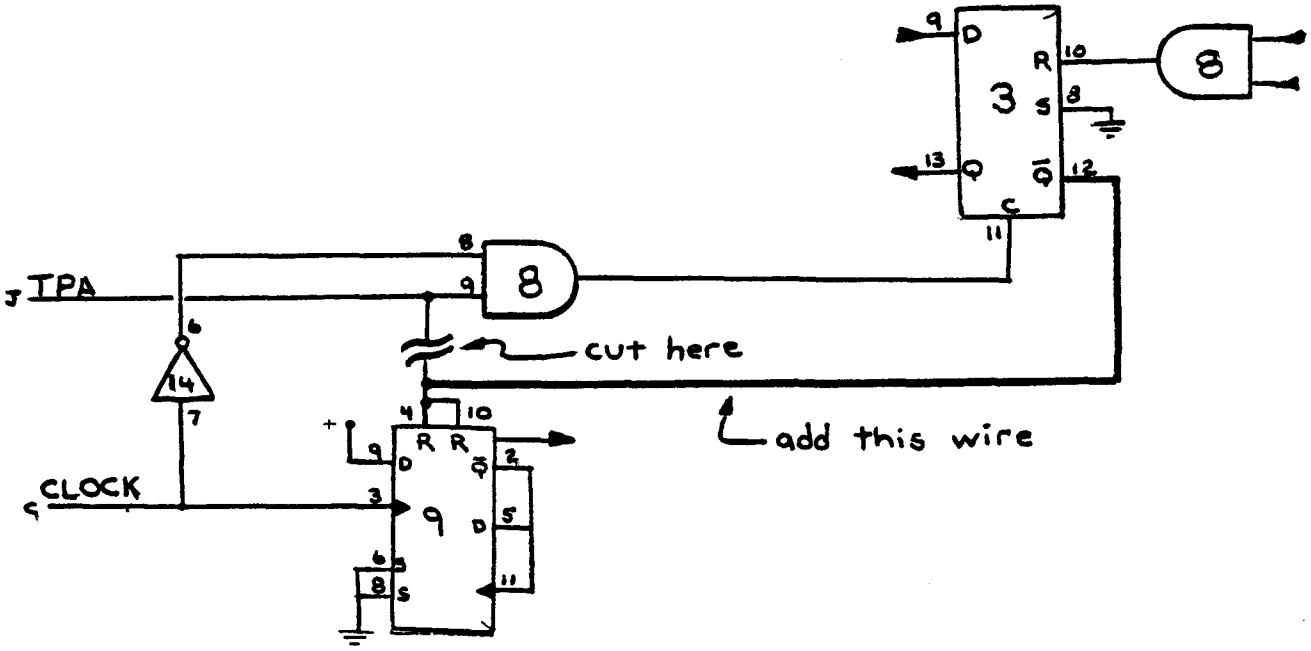
HARDWARE FIXES FOR THE ACE DISK CONTROLLER BOARD

by- T Hill 30-481 Pitfield Rd Milton Ontario

The ACE disk controller board has been available for over a year now. Quite a number of these boards are in service (including the one at my house). However there is one artwork problem with the board that has not been reported until now, and one shakey timing condition.

The artwork problem with the board has to do with a short circuit between pins 5 and 6 of U17. There is a connection on the bottom of the board that should not be there, due to one of the traces being too close to the pins. A little circuit board surgery with a sharp knife and some jumper wire should solve the problem. Pin 5 should NOT be connected to the offending trace, while pin 6 should. My board worked fine without this fix, until I tried to access the inner disk tracks. Then it was game over for the data I had tried to store there.

The timing problem with the board may or may not be a problem with any particular system, depending on how heavily loaded TPA is and on the system clock frequency. My board worked fine at a clock speed of 1.78 MHz but went wierd at 2.00 MHz. The disk would do a Master Reset no matter what command was given. I traced the problem to the timing relationship between TPA and the system clock. Basically, as TPA gets more loaded, it takes longer to become high with respect to the system clock. The fix shown below gets around the problems created by this delay.



FULL SCREEN EDITOR FOR Fig-FORTH

by- T Nery 33 County St. , Foxboro, Ma. USA 02035

SCR # 2

```
 0     ( SCR # 2: Fig-FORTH EDITOR Background ) -->
 1
 2         The following editor gives any computer system using a
 3     Neutronic's old style video display board with FORTH, full
 4     screen edit capability.
 5
 6         This editor uses the full 64 by 16 charater display, as set
 7     FORTH (sic.) in the fig-FORTH model.
 8
 9         While it possible to adapt this program to any video
10     display unit, the modifier should be aware of the use of charac-
11     ter timing on some special functions, such as EOL.  Other than
12     these the only routine requiring modification is the definition
13     ' CURPOS ' which positions the cursor using the Neutronic's
14     escape sequence.
15
```

SCR # 3

```
 0     ( SCR # 3: User instructions) -->
 1     Screen selection:    n1 CLEAR - clears n1 screen and selects it
 2                          n1 LIST  - lists  n1 screen and selects it
 3     Load Mode:           KBLOAD   - selected screen loaded w/ 16 line
 4     Edit Mode:           EDIT     - selected screen put in edit mode
 5                                     - terminated with escape key
 6     Edit Mode Commands:  ^S       - move cursor one space left
 7                          ^F       - move cursor one space right
 8                          ^E       - move cursor up one line
 9                          ^C       - move cursor down one line
10                          ^I       - Insert mode, type char. to be
11                                     inserted at cursor follow w/ cr
12                          ^K       - Delete mode, type over char. to
13                                     be deleted followed w/ cr
14                          ^M       - Modify mode, type over char. to
15                                     be replaced followed w/ cr
```

SCR # 4

```
 0     ( SCR # 4: User instructions, cont. )   -->
 1                          ^O       - Open cursor line by moving cursor
 2                                     line down. Old line 16 is lost.
 3                           ^L      - Delete cursor line by moving all
 4                                     lines below cursor up. Line 16
 5                                     will be blank.
 6     Screen Command       n1 n2 COPY
 7                                     - Copy screen n1 to n2. Screen n1
 8                                     remains intact.
 9
10     FORTH words required (other than standard distribution):
11
12     INDEX --> LOAD MESSAGE .LINE (LINE) BLOCK BUFFER R/W HI LO DR1
13     DRO EMPTY-BUFFERS UPDATE +BUF PREV USE
14
15     from the installation manual.
```

SCR # 5

```
 0 ( SCR # 5: PICK, CUR-X, CUR-Y, DELTA )
 1 FORTH DEFINITIONS HEX
 2 : PICK
 3    2 * SP@ SWAP - @ ;
 4
 5 VOCABULARY EDITOR IMMEDIATE
 6
 7 : WHERE
 8    DUP B/SCR / DUP SCR ! ." SCR # " DECIMAL .
 9    SWAP C/L /MOD C/L * ROT BLOCK + CR C/L TYPE
10    CR HERE C@ - 3 + SPACES 5E EMIT [COMPILE] EDITOR QUIT ;
11
12 EDITOR DEFINITIONS HEX
13
14 0 VARIABLE CUR-X    0 VARIABLE CUR-Y    0 VARIABLE DELTA
15 3C00 CONSTANT SCR#0  -->
```

SCR # 6

```
 0 ( SCR # 6: CURSOR CONTROLS )
 1 : CURMOVE
 2    3D 1B 4 0 DO EMIT LOOP ;
 3
 4 : CXYMOVE
 5    CUR-X @ CUR-Y @ CURMOVE ;
 6
 7 : CYOMOVE      0 CUR-Y @ CURMOVE ;
 8 : CFMOVE       2C SWAP CURMOVE ;
 9
10 : XINC         CUR-X @ 1+  CUR-X ! ;
11 : XDEC         CUR-X @ 1 - CUR-X ! ;
12 : YINC         CUR-Y @ 1+  CUR-Y ! ;
13 : YDEC         CUR-Y @ 1 - CUR-Y ! ;
14 -->
15
```

SCR # 7

```
 0 ( SCR # 7: STRING COMMANDS AND SCREEN CONTROLS )
 1 : <CMOVE
 2    DUP ROT + SWAP ROT 1 - DUP ROT +
 3   DO 1 - I C@ OVER C! -1 +LOOP DROP ;
 4
 5 : CMOVE$      OVER 4 PICK > IF <CMOVE ELSE CMOVE ENDIF ;
 6
 7 : LEN
 8    255 0 DO DUP I + C@ 0= IF I LEAVE ENDIF LOOP
 9    SWAP DROP ;
10
11 : DELAY       BEGIN 1 - -DUP 0= UNTIL ;
12
13 : CLS        C EMIT 100 DELAY ;
14 : EOLS        5 EMIT ;
15 -->
```

SCR # 8

```
 0 ( SCR # 8  )
 1 : CFLD
 2    1 CFMOVE EOLS 100 DELAY
 3    37 1 CURMOVE ." SCR # " SCR @ .
 4    0 CFMOVE EOLS ;
 5
 6 : LEDIT       CUR-Y @ 40 * SCR#0 + ;
 7 : CEDIT       LEDIT CUR-X @ + ;
 8
 9 : ETYPE       0 CFMOVE ;
10 : CTYPE       ETYPE EOLS 100 DELAY CXYMOVE ;
11
12 : LPR         LEDIT 40 -TRAILING TYPE DELTA @ SPACES ;
13 : ELPR        CYOMOVE LPR CTYPE ;
14 -->
15
```

SCR # 9

```
 0     ( SCR # 9: LINES, PELPR, COPY )
 1     : LINES
 2        10 SWAP DO CR
 3                  I 40 * SCR#0 + 40 -TRAILING TYPE
 4                LOOP ;
 5
 6     : PELPR       CUR-Y @ DUP 0=
 7        IF CLS
 8        ELSE YDEC CYOMOVE 6 EMIT YINC
 9             F CUR-Y @ - 25 * DELAY
10        ENDIF
11        LINES CTYPE CXYMOVE ;
12
13     : COPY
14        400 * SCR#0 + SWAP 400 * SCR#0 + SWAP 400 CMOVE ;
15     -->
```

SCR # 10

```
 0     ( SCR #10: LIST, CLEAR )
 1     : LIST
 2        DUP SCR !
 3        0 COPY
 4        CLS 0 LINES CFLD ;
 5
 6     : CLEAR
 7        SCR !
 8        SCR#0 400 BLANKS
 9        CLS CFLD ;
10     -->
11
12
13
14
15
```

SCR # 11

```
 0     ( SCR #11: LOCHK, LFCHK, CNT-E, CNT-C )
 1     : LOCHK       CUR-Y @ 0= ;
 2     : LFCHK       CUR-Y @ F = ;
 3
 4     : CNT-E
 5        LOCHK
 6          IF F CUR-Y ! CXYMOVE
 7          ELSE YDEC B EMIT ENDIF ;
 8
 9     : CNT-C
10        LFCHK
11          IF 0 CUR-Y ! CXYMOVE
12          ELSE YINC A EMIT ENDIF ;
13     -->
14
15
```

SCR # 12

```
 0     ( SCR #12: CNT-F, CNT-S )
 1     : CNT-F
 2        CUR-X @ 3F =
 3          IF LFCHK
 4             IF 0 CUR-X ! 0 CUR-Y !
 5             ELSE YINC 0 CUR-X ! ENDIF CXYMOVE
 6          ELSE XINC 9 EMIT ENDIF ;
 7
 8     : CNT-S
 9        CUR-X @ 0=
10          IF LOCHK
11            IF F CUR-Y !
12            ELSE YDEC
13            ENDIF 3F CUR-X ! CXYMOVE
14          ELSE XDEC 8 EMIT ENDIF ;
15     -->
```

SCR # 14

```
0     ( SCR #14: REPLACE, DELETE )
1
2     : REPLACE      CXYMOVE
3        PAD 40 CUR-X @ - EXPECT
4        PAD CEDIT PAD LEN DUP CUR-X @ + CUR-X ! CMOVE$
5        0 DELTA ! ELPR ;
6
7     : DELETE       CXYMOVE
8        PAD 40 CUR-X @ - EXPECT
9        PAD LEN DELTA !
10       CEDIT DUP DELTA @ + SWAP  40 CUR-X @ - DELTA @ - CMOVE$
11       LEDIT 3F + DELTA @ - DELTA @ BLANKS
12       ELPR ;
13    -->
14
15
```

SCR # 15

```
0     ( SCR #15: INSERT )
1     : INSERT     CXYMOVE
2        PAD 40 CUR-X @ - EXPECT
3        PAD LEN DELTA !
4        CEDIT DUP DELTA @ + 40 CUR-X @ - DELTA @ - CMOVE$
5        PAD CEDIT DELTA @ CMOVE
6        CUR-X @ DELTA @ + CUR-X !
7        0 DELTA ! ELPR ;
8     -->
9
10
11
12
13
14
15
```

SCR # 13

```
0     ( SCR #13: FLUSH, KBLOAD, COMPRESS, SPREAD )
1     : FLUSH      0 SCR @ COPY ;
2
3     : KBLOAD     0 0 CURMOVE
4        SCR#0 DUP 400 + SWAP DO I 40 EXPECT CR 40 +LOOP FLUSH ;
5                    +
6     : COMPRESS     CXYMOVE
7        LEDIT DUP 40 + SWAP   F CUR-Y @ - 40 * CMOVE$
8        SCR#0 3C0 + 40 BLANKS        0 DELTA ! PELPR ;
9
10    : SPREAD       CXYMOVE
11       LEDIT DUP DUP 40 +    F CUR-Y @ - 40 * CMOVE$
12       40 BLANKS            0 DELTA ! PELPR ;
13    -->
14
15
```

SCR # 16

```
0     ( SCR #16: EDIT )
1     : EDIT     0 CUR-X !  0 CUR-Y ! CXYMOVE
2        BEGIN
3          KEY
4             DUP  5 = IF CNT-E ENDIF
5             DUP  3 = IF CNT-C ENDIF
6             DUP 13 = IF CNT-S ENDIF
7             DUP  6 = IF CNT-F ENDIF
8             DUP  9 = IF ETYPE ." INSERT "   INSERT   ENDIF
9             DUP  B = IF ETYPE ." DELETE "   DELETE   ENDIF
10            DUP  D = IF ETYPE ." REPLACE "  REPLACE  ENDIF
11            DUP  F = IF ETYPE ." SPREAD "   SPREAD   ENDIF
12            DUP  C = IF ETYPE ." COMPRESS " COMPRESS ENDIF
13         1B =
14       UNTIL FLUSH EMPTY-BUFFERS ;
15    ;S
```

FORTH FULL SCREEN EDITOR AND MORE

by - D Hall, 4565 E. Industrial St. 7-K, Simi Valley, Ca. 93063

I have followed the series of articles concerning FORTH, and its implementation on RCA 1802 based systems with a great deal of interest. My 1802 system consists of a modified Quest Super Elf, with the S-100 expansion option installed. I am currently using two S-100 boards; a 64X16 memory mapped video board, and a 64K static ram board. I have been using S. Nies' first monitor (IPSO FACTO #16) for some time now and have finally gotten fig-FORTH up and running. I have installed the various "fixes" printed in IPSO FACTO, and done a little work of my own.

In All About Forth by Glen Haydon, and Starting Forth by Leo Brodie, there are two words in FORTH-79 used to list the contents of the parameter stack non-destructively. DEPTH and .S as they are shown will not work properly on a fig-FORTH system, but the words can be made to work. I have found .S to be very useful when debugging new words. Simply enter ".S" followed by a carriage return, and the system responds by printing all the values currently on the parameter stack, without changing any of them. All values are printed in the current number base, as unsigned 16 bit numbers.

```
: DEPTH          SP@ S0 @ - 2 / ;

: .S             CR DEPTH
                 IF SP@ S0 @ DO I 2+ @ U. 2 +LOOP
                 ELSE ." the Stack is Empty "
                 THEN ;
```

My work involves creating the operating software for micro-processor based systems used in TeleCommunications applications. Our developement systems use the Z-80 and 8085 micro-processors; therefore I have implemented fig-FORTH on these systems as well as my ELF system. Finally, I also own a TRS-80 Model I, and I have ESF FORTH for it.

With my ESF FORTH system, I got a screen editor. I have modified that editor to work on my ELF. I am not sure whether this is copyrighted material, and since EXATRON is no longer in the home computerist business, I have been unable to learn anything from them. I am enclosing a listing in any case. As written, the screen editor is set up for a 64X16 display format, and it is assumed that the video is memory mapped, and that control codes can be entered from the system keyboard.

By entering "nn EDIT" carriage return, screen #nn will be displayed on the video display, ready for editing. Once control is passed to EDIT, the only ways to get out are to reset the computer, or press the ESC key. If exit from EDIT is effected by pressing ESC, the screen just referenced is marked as updated. Control codes are used to implement the insert (both character and line) and delete (char/line) functions, as well as the cursor position control functions. With the exception of the form feed function (CTL/L), all the cursor control functions are non-destructive. All codes that are not defined for special function use are "displayed" (or they can ring the "bell").

CTL/F	advance cursor 1 space.
CTL/H	back cursor 1 space.
CTL/I	tab cursor to next tab stop (tab stops every 8 spaces)
CTL/J	downward line feed.
CTL/K	upward line feed.
CTL/L	clear the screen and home the cursor.
CTL/M	carriage return/line feed
RUBOUT	back cursor 1 space.
CTL/Q	insert a blank line on the cursor line. (line 15 is lost)
CTL/R	delete line on cursor line. (line 15 is blanked)
CTL/S	insert a blank character at the cursor position. (last character of line is lost)
CTL/T	delete the character at the cursor position. (last character of line is a blank)

Finally, I found the way VLIST displayed the words in the dictionary was uncomfortable for me to try to read. The listing of the improved VLIST checks for line wrap-around before typing the word name.

```
: VLIST
        CR CR 0 OUT ! CONTEXT @ @
        BEGIN DUP DUP 1+ SWAP C@ 1F AND DUP 3 + OUT @ + C/L >
              IF CR 0 OUT ! THEN
              TYPE SPACE SPACE PFA LFA @ DUP
              0= ?TERMINAL OR UNTIL DROP CR ;
```

```
SCR # 6
  0 ( Screen Editor                                                  )
  1         FORTH DEFINITIONS HEX
  2
  3 9880 CONSTANT CURSOR        ( Cursor Storage Address             )
  4 A000 CONSTANT VRAM          ( Start 1st Line Video Display       )
  5 A380 CONSTANT VTOP          ( Start Last Line Video Display      )
  6   3F CONSTANT C/L-1         ( Length of Video Line - 1           )
  7    0 VARIABLE TEMP-CHAR     ( Temp Storage Char @ Cursor         )
  8
  9 : LINE-ADDR
 10          CURSOR @ FFC0 AND ;
 11 : LINE-NUM
 12          LINE-ADDR VRAM - C/L / ;
 13 : CLS
 14          0C EMIT ;
 15 -->


SCR # 7
  0 ( Screen Editor                                                  )
  1
  2 : CURSOR-ON
  3          CURSOR @ C@ TEMP-CHAR C! 7F CURSOR @ C! ;
  4 : CURSOR-OFF
  5          TEMP-CHAR C@ CURSOR @ C! ;
  6 : LINE-INS
  7          CURSOR-OFF VTOP 0F LINE-NUM
  8          DO DUP DUP C/L + C/L CMOVE C/L - LOOP
  9          C/L + C/L BLANKS CURSOR-ON ;
 10 : LINE-DEL
 11          LINE-NUM 0F < IF
 12          CURSOR-OFF LINE-ADDR 0F LINE-NUM
 13          DO DUP DUP C/L + SWAP C/L CMOVE C/L + LOOP
 14          C/L BLANKS CURSOR-ON ENDIF ;
 15 -->


SCR # 8
  0 ( Screen Editor                                                  )
  1 : CHAR-DEL
  2          CURSOR-OFF LINE-ADDR C/L-1 + DUP CURSOR @
  3          DO I 1+ C@ I C! LOOP
  4          BL SWAP C! CURSOR-ON ;
  5 : CHAR-INS
  6          CURSOR-OFF CURSOR @ LINE-ADDR C/L-1 +
  7          DO I 1 - C@ I C! -1 +LOOP
  8          BL CURSOR @ C! CURSOR-ON ;
  9 : PRE-CURSOR
 10          CURSOR @ DUP FC00 AND SWAP ;
 11 : POST-CURSOR
 12          3FF AND OR CURSOR ! ;
 13 CREATE (BEL)
 14          F8 C, 80 C, A7 C, D4 C, 88 C, ED C, DC C, SMUDGE
 15 -->
```

```
SCR # 9
  0 ( Screen Editor                                                  )
  1 : (BS)
  2         PRE-CURSOR 1 - POST-CURSOR ;
  3 : (HT)
  4         PRE-CURSOR FFF8 AND 8 + POST-CURSOR ;
  5 : (LF)
  6         PRE-CURSOR C/L + POST-CURSOR ;
  7 : (VT)
  8         PRE-CURSOR C/L - POST-CURSOR ;
  9 : (FF)
 10         CLS BL CURSOR @ C! ;
 11 : (CR)
 12         PRE-CURSOR FFC0 AND C/L + POST-CURSOR ;
 13 : (ACK)
 14         PRE-CURSOR 1+ POST-CURSOR ;
 15 -->


SCR # 10
  0 ( Screen Editor                                                  )
  1
  2 : SEMIT    CURSOR-OFF
  3 (  CTL/F  )      DUP 06 = IF DROP (ACK) ELSE
  4 (  CTL/H  )      DUP 08 = IF DROP (BS)  ELSE
  5 (  CTL/I  )      DUP 09 = IF DROP (HT)  ELSE
  6 (  CTL/J  )      DUP 0A = IF DROP (LF)  ELSE
  7 (  CTL/K  )      DUP 0B = IF DROP (VT)  ELSE
  8 (  CTL/L  )      DUP 0C = IF DROP (FF)  ELSE
  9 (  CTL/M  )      DUP 0D = IF DROP (CR)  ELSE
 10 ( RUB-OUT )      DUP 7F = IF DROP (BS)  ELSE
 11                  DUP BL < IF DROP (BEL) ELSE
 12                  CURSOR @ C! (ACK)
 13             THEN THEN THEN THEN THEN THEN THEN THEN THEN
 14             CURSOR-ON ;
 15 -->


SCR # 11
  0 ( Screen Editor                                                  )
  1 : EDIT         DUP SCR ! BLOCK VRAM B/BUF CMOVE VRAM CURSOR !
  2         CURSOR-ON BEGIN KEY DUP 1B = 0= WHILE      ( ESC TO END )
  3
  4 ( CTL/Q )      DUP 11 = IF DROP LINE-INS ELSE
  5 ( CTL/R )      DUP 12 = IF DROP LINE-DEL ELSE
  6 ( CTL/S )      DUP 13 = IF DROP CHAR-INS ELSE
  7 ( CTL/T )      DUP 14 = IF DROP CHAR-DEL ELSE
  8                SEMIT
  9
 10                ENDIF ENDIF ENDIF ENDIF
 11         REPEAT DROP CURSOR-OFF
 12         VRAM SCR @ BLOCK B/BUF CMOVE UPDATE
 13         CLS CURSOR-ON CR ;
 14
 15
```

Mailing Label Record Program

One day last month one of my friends stopped by and asked if I would be kind enough to run off a few mailing labels. He wanted to notify all of his relatives of the upcoming family reunion. "That should be easy with a printer and plenty of labels," he said. As I did not care to type all of those names and addresses myself, I created this program and let him enter all of those names and addresses while I read the latest issue of Microcomputing. Since then, I have used it for several mailing lists.

The program allows you to create a file from scratch, then edit it later. As most people change addresses or phone numbers once in a while, an edit of some kind is required. To edit a record, you must first find it, so I added a search mode. Every once in a while it would be nice to delete a record without leaving a blank space in the file in its place. In order to do this I added a delete mode. Using the DLOAD and DSAVE commands allow saving a file for later use as needed. The last mode needed is a print routine to actually print the labels. It would be nice to have a hard copy of the file to edit as needed. I added this print mode in addition to just printing the labels on paper.

This program is written in Quest Basic V5.0, using an ADDS Viewpoint terminal and an Epson MX=80 printer. The ADDS has a cursor positioning mode and a half brightness mode in addition to other features. In order to understand

the code in the first half of the program I need to explain how the cursor positioning works. This terminal requires a four character sequence to put the cursor anywhere on the screen. This is ESC Y r c where r and c are the required row and column numbers. The easy way to work this out is to use (31+(r or c)) where r is 1 to 24 and c is 1 to 80.

For the upper left hand corner this is ESC Y SPACE SPACE, which is next to impossible to use with basic. The CHR$ function provides a very easy way to get a handle on this type of thing. Use CHR$(27,89,(31+1),(31+1));, this is ESC Y SPACE SPACE in decimal format. CHE$(27,107) is a clear to lower right hand corner from the current position.

The last terminal control code is in three parts. The first part requires that the half brightness mode be set first, then turned on and off as needed. CHR$(27,48,65) sets the half bright mode. CHR$(14) turns on half bright mode and CHR$(15) turns it off. These two CHR$ commands are used in three lines.

In the printer routines, two CHR$ commands are used. These set and reset what Epson calls overstrike and emphasized. CHR$(27,69,27,71) turns this print mode on while CHR$(27,70,27,72) turns it off. CHR$(10) is just the same as a blank PRINT statement but requires less memory space to store it.

Now that I have described the strange stuff that my hardware uses in a way that will allow you to modify the code for what ever you are using, I'll get on to the real program.

Line 40

This sets up the terminal for the 1/2 brightness mode.

Line 50

This sets M$ up for use later as a file title. A and N are set to zero. A is used as the current record counter. N is used in the print routines.

CLS just clears the screen.

Line 60

This line positions the cursor to the beginning of line 11 and clears to the end of the screen. As this is a short subroutine, it returns to the calling line.

Line 70

Position the cursor to row 13, column 20, then return.

Line 80

This is used to print three spaces at row 1, column 7. It then prints the current record number starting at the same place. This is also a subroutine.

Line 90

Prints RECORD in the upper-left corner of the screen and the words BYTES REMAINING at the upper right corner, leaving room for what the MEM command prints later. Lets you keep track of how much memory is left the easy way.

Lines 100 through 160

These lines print a header for each of the seven lines of text starting at line four. Change these as desired. This is the only time this section of code is run. As only the first four lines of text will print on labels, I recommend that you use either the provided text or something like it. All seven will print out in the plain paper mode. Great for phone numbers and other things.

Line 170

This calls clear to end of screen then prints three blank lines.

Lines 180 through 230

Print all of the command functions of the program.

Line 240

Positions the cursor, prints prompt, and clears to the end of the screen.

Lines 250 through 320

Check for valid entry and go to the desired routine. Function 1 sets A to one to index into the array used for storage before going to the create routine.

Lines 330 through 350

This is a subroutine that prints the dots for the data entry field. The line to print is determined by adding the variable "L" to 35 to give the required row number.

The column is the same for each row (23), leaving the line titles intact. The "14" in the first CHR$ turns on the half brightness mode while the "15" in the other CHR$ turns it off. This places one dim dot for each of the 32 possible characters on each line. Before this subroutine returns it also positions the cursor in the correct place for the MEM function results.

Lines 360 through 400

This subroutine is used for the actual data entry. Line 360 places the cursor to the start of each of the seven rows of dots, as determined by the value in L. The data is input for the line by the INPUT statement. Line 370 and 380 checks for over 32 characters. If this happens, the dots are redrawn over the entered text and LINE TOO LONG is printed to the right. The routine will then go

back to 360 for another input.

Line 390 will create a single space if only a carriage is entered. This makes the search routine code a lot easier to work with and insures at least one character is entered for every string entered.

Line 400 just prints the previously entered data. It then clears to the end of the current line removing all remaining dots not overwritten, as well as erasing the LINE TOO LONG message, if printed. The MEM command is again called, updating the bytes remaining data field before returning. You MUST quit entering data if this gets to less than 256 bytes or the program, and possibly the Basic, crashes. If this occurs, you WILL loose all data entered.

Lines 410 through 470

This routine will print out the seven lines of the current record before returning.

Lines 480 through 510

This subroutine is called by most of the modes to verify that the current data is correct. If the correct reply is not given, the routine will loop until a correct response is given.

Lines 520 through 630

This section of code asks for the line number where any incorrect data has been input. GOSUB 70 positions the cursor before the INPUT statement is executed. GOSUB 60 clears from the start of the current line to the end of the screen. Most of the INPUT lines use this to keep the screen clear of unwanted characters. Range checking is taken care of with lines 530 and 540.

Line 550 will print a row of dim dots for the required

line using "L" provided by line 520. The data input at line 360 is used for the actual input.

Lines 560 through 620 transfer the current contents of Z$ to the correct string variable for use later.

Line 630 reprints the current data while clearing any remaining dots or error messages before returning.

Lines 680 through 760

This subroutine transfers the contents of one line of the current record into Z$, used as a temporary buffer for later use. GOSUB 360 is called for the data input.

Line 770

This subroutine transfers the contents of the current record into a temporary one record buffer, then returns.

Line 780

This routine does the reverse of the above routine.

Lines 790 and 800

This subroutine creates one additional label record that is all blanks. This is used for the delete mode and, if needed, in the print routines.

Lines 810 through 880

This is the actual create mode code. Almost all of this code causes calls to the previously described routines.

GOSUB 330 draws the dots for each of the seven lines, GOSUB 80 prints the current record number and GOSUB 680 gets the data to be input. 680 calls 360 seven times to get a complete record before returning to line 820.

GOSUB 410 reprints the complete record. GOSUB 480 asks if the entered data is correct. If 480 returns with Z$ equal to "N" then GOSUB 520 is called. This allows one line of text to be corrected. The program will then loop back to line 830, again asking if the record is correct.

If GOSUB 480 returns with Z$="Y", then GOSUB 770 is called, transferring the complete temporary record to the storage array area. The variable "A" is used to index into the array. One is now added to A, setting A up to point to the area used to store the next record. GOSUB 640 is now called, asking if another record is to be entered. If this subroutine returns with Z$="Y", the program will loop back to line 810 for the next record entry.

If Z$="N", then the program will return to the function display in the lower half of the screen.

Lines 890 through 940

This is the data save mode code. GOSUB 70 positions the cursor for the DATA SAVE mode message. Line 900 is self explaining except for M$. M$ is used for a title and date of the record to be saved. As this is not used in any way by the code, you may use it for anything desired.

Line 910 lets you enter a new title or keep the old data. If you desire to keep the old one enter ONE space and a carriage return. Lines 920 and 930 print a prompt for you to set up the tape recorder and save the data to tape. After saving the data the program loops back to the mode menu.

Lines 950 through 1000

In this section is the code that allows you to load a tape that was created and saved previously. Line 950 calls the clear to end of screen then prints the DATA LOAD MODE message. Lines 960 and 970 work the same as lines 930 and 940 in the save mode except that data is loaded from the tape. After DLOAD C completes, line 980

will print the last updated message along with whatever M$ has in it. Line 990 is self explaining. Line 1000 loops back to the mode menu.

Lines 1010 through 1370

This area contains the edit routines. Two basic edit modes are provided. One will allow addition to the file that already exists and the other provides a method of finding and editing a record that currently exists.

Lines 1010 through 1040 select between these two modes of editing. If the ADD mode is selected, one is added to the current value of A, then the program simply goes to the create mode code. 1020 and 1030 provide a means of error checking the selected command.

If the search mode is selected, line 1040 sets A to zero before going into the search code. GOSUB 330 is then used to redraw the dots, clearing out any data left from the last record. Line 1050 will first position the cursor, then ask for what line the data to search for is on. The requested line will be saved in variable L. GOSUB 60 clears out the bottom of the screen. Lines 1060 and 1070 provide the range checking. Line 1080 calls GOSUB 360, placing the question mark on the requested line.

The search routine will allow any length of data to search on up to the full 32 characters. The input sub-routine at 330 limits the input to 32 characters. For a search, both the input data and the record MUST be the same for EVERY character. Line 1090 will set T equal to the number of characters in the search data. Line 1100 increments A to index through the records. Line 1110 checks if A (current record number) is greater than A1

(total number of records). If this occurs, the record not found message is output, followed by a branch to the search for another record question on line 1340.

If the current record number is less than or equal to the total record count, lines 1120 through 1180 will extract a search string equal in length to the search data length. Variable L from 1050 will determine which lines of the records are searched. Line 1190 does the actual checking for the match. Because the MID$ command requires at least a one for the starting character pointer to work correctly, there cannot be any "blank" lines in the data array. The forced "space" provided in the subroutine at line 360 takes care of this. Therefore, you will always have at least MID$(X$(A),1,1).

If this record does not match, the program loops back to line 1100 to check the next record. If there is a match, line 1200 is executed.

Line 1200 calls GOSUB 780, copying the entire record out to the temporary storage area for one record. GOSUB 410 will then print the entire record on the screen. GOSUB 80 prints the record number in the correct place on the screen. In case this is not the desired record, the search string is saved in Z$.

Line 1220 is now executed, returning either "Y" or "N". If z$="N", then Z$ is restored to the search string data. Now the next record in the array is checked. If the record is the desired one, a delete option is provided now.

If delete is selected, line 1280 calls the subroutine at line 790 to create a "blank" record at the end of the current record array.

30

As the record counter (A1) is increased by one, this counter must be decreased by one, to restore the count to the "true" record count.

Lines 1290, 1300, and the first two sections of line 1310 form a FOR NEXT loop that first copies all seven lines of the next record into the current record. The GOSUB 80 in this loop just prints the number of the record being deleted. This will increment once for each pass through the loop.

After the loop finishes, the total record count needs to be decreased by one, to reflect that there is one less record in the array. The program will then go to line 1340 to ask if you want to search for another record.

If the delete option was not selected, line 1320 will be the next line executed. GOSUB 480 is the routine that asks if the current record is correct. If this subroutine returns with Z$ equal to "N", line 1330 is executed. GOSUB 520 asks what line is wrong, then calls GOSUB 360 (the actual data input routine), prints the corrected data, then returns to the rest of line 1330. The second section of line 1330 calls GOSUB 770 to save the corrected data in the record array. The last part of the line is a branch back to 1320, asking again if the record is correct.

If GOSUB 480 returns with Z$="Y", line 1340 is the next line executed. This gives you the option of searching for another record or returning to the function menu. If search for another record is selected, the program will branch back to line 1040, starting the edit routines over.

Lines 1380 through 1640

These lines contain the printer routines for both the labels and paper printout. Lines 1380 through 1400 select one of the two print modes. The label mode will print only the first four lines using the combined over-strike and emphasized Epson print modes. In the plain paper mode, all seven lines of text are printed using the standard one pass print mode.

Most of this section is used by both modes. Variable T is used as the mode select flag while L is used as counter in the plain paper mode. The routines are set up for 2 up, 15/16 by 3 1/2 labels. If different labels are used, this routine will have to be reworked.

If the label mode is selected, the variable T will be set to one before going to line 1420. This line gives you a chance to set the labels in the correct place before the actual printing starts. GOSUB 70 and GOSUB 60 actions have been described in the above routines. Line 1430 switches all output to the printer. The plain paper mode works the same way, excetp T will be zero.

As these routines print both the left and right labels together, one line at a time, there MUST be an even number of labels or you will get an undefined string error when the last labels start to print. Line 1440 checks for an even number of labels. If the record count in A1 is even, the program will branch to 1460. In the odd case, GOSUB 790 is called, creating one "blank" label at the end of the record array. N is set to one as a flag, so this "extra" label can be removed after the printing is finished.

Line 1460 sets up a counter used in the plain paper

print mode to count how many records are printed on a sheet of paper. This is not used in the label print mode.

Line 1470 is used to select the kind of type used to print with. If T is one, labels are desired, so set the high quality print mode, otherwise leave the printer in normal print.

Lines 1480 through 1610 is the actual print code. This is mostly a FOR NEXT loop. If plain paper is selected (T=0), and this is the first line of the paper, do two line feeds to create a top margin. Next, tab over to column 20 and print M$ (the file date or whatever) as a header, then line feed two more times.

Lines 1500 through 1530 are always executed to give the first four lines of the record. Line 1540 checks if the print desired is labels. If labels are being printed this time, two line feeds are needed to get to the next label. This is done in line 1590. If paper is being used, the last three lines need to be printed. Lines 1550 through 1570 do this. As this is a STEP 2 loop, the right side of each print line needs to be current value of A plus one or (A+1) to print the correct record.

Line 1580 gives a single line feed. If the paper mode is selected, three blank lines will be printed before the next pair of records are printed. This gives twelve records of seven lines each on a page, as line 1590 is always done.

Line 1600 is used by the paper mode (T=0) to keep count of how many records are printed on a page. If six pairs have been printed, including the three blank lines between pairs, a top-of-form is issued to get to the top

of the next page. Line 1610 is the end of the FOR NEXT loop.

As all records have been printed now, it would be nice to leave the printer set to normal print if labels were printed. Line 1620 does this by cancelling the two print modes set, if labels were printed.

Line 1630 does two things. The first thing is to set all output back to the terminal. The second thing done is to remove the extra "blank" label added if there was an odd number of records. The variable N will be one if an extra record was created.

Line 1640 just goes and prints the mode select menu again.

There are several other things about this program not covered in the code description. One is that you CANNOT have over 254 records at a time to work with due to the fact that the Quest basic only supports string arrays from X$(1) to X$(255). As the delete routine requires one additional variable to work, you MUST not have more than 254 records at a time. More records than this will have to be done in groups. All terminal output is designed for an 80 by 24 terminal, therefore the screen layout will have to be redone for other screen formats.

The other thing needed is a sort routine. This can be done as part of this program, but to allow more space for records, this is not in the program. Because Quest basic allows data created by one program to be used by another program, a sort routine can be a separate program.

If mail lists over 254 records are to be used, it is not possible to print all labels at one time. The solution is

to use a separate program that reads a file, prints the labels, then reads the next file and so on. This can be done very easily by using a FOR NEXT loop set to the number of files desired. Inside the loop, use a DLOAD,C followed by a modified version of the current print routines. I use this type of thing if I have text that runs over 248 lines created on a slightly different version of the text editor described by Fred Hannon in Questdata volume 3, issue #2. As the number of lines you can get at a time is limited to 255 for the reasons described above, I make my text in groups of 248 lines. This gives a nice break at the end of the last page. Extra lines just end anywhere on the last page.

```
10 REM MAILING LABEL PROGRAM VER 1.7  10/16/83
20 REM  Written by     John Ware,  2257 6th. Ave.
30 REM                 Ft. Worth,  Tx.  76110
40 PRINT CHR$(27,48,65);
50 M$="*** NO DATE (NEW FILE) ***":A=0:N=0: CLS: GOTO 90
60 PRINT CHR$(27,89,43,32,27,107): RETURN
70 PRINT CHR$(27,89,45,52);: RETURN
80 PRINT CHR$(27,89,32,39);"   ";CHR$(27,89,32,39);A: RETURN
90 PRINT CHR$(27,89,32,32);"RECORD";CHR$(27,89,32,82);"BYTES REMAINING"
100 PRINT CHR$(27,89,36,32);"LINE 1  (Name)"
110 PRINT CHR$(27,89,37,32);"LINE 2  (Address)"
120 PRINT CHR$(27,89,38,32);"LINE 3  (City & State)"
130 PRINT CHR$(27,89,39,32);"LINE 4  (Zip)"
140 PRINT CHR$(27,89,40,32);"LINE 5  (Ph. No.)"
150 PRINT CHR$(27,89,41,32);"LINE 6  (Comments)"
160 PRINT CHR$(27,89,42,32);"LINE 7  (Comments)"
170 GOSUB 60: PRINT CHR$(10,10,10)
180 PRINT TAB(10);"CREATE NEW FILE ---------------- 1"
190 PRINT TAB(10);"EDIT A RECORD ------------------ 2"
200 PRINT TAB(10);"SAVE FILE TO TAPE -------------- 3"
210 PRINT TAB(10);"LOAD A FILE FROM TAPE ---------- 4"
220 PRINT TAB(10);"PRINT FILES -------------------- 5"
230 PRINT TAB(10);"QUIT --------------------------- 6"
240 PRINT CHR$(27,89,54,42);: INPUT "ENTER NUMBER OF DESIRED FUNCTION "X: GOSUB
60
250 IF X<1 GOTO 240
260 IF X=1A=1: GOTO 810
270 IF X=2 GOTO 1010
280 IF X=3 GOTO 890
290 IF X=4 GOTO 950
300 IF X=5 GOTO 1380
310 IF X=6 PRINT CHR$(27,0): END
320 IF X>6 GOTO 240
330 FOR L=1 TO 7
340 PRINT CHR$(27,89,(35+L),55,14);".............................";CHR$(15):
NEXT
350 PRINT CHR$(27,89,32,98);: MEM : RETURN
360 PRINT CHR$(27,89,(35+L),54);: INPUT Z$
370 IF LEN(Z$)>32 PRINT CHR$(27,89,(35+L),54,14);" .....................
...";CHR$(15);
380 IF LEN(Z$)>32 PRINT "  LINE TOO LONG": GOTO 360
390 IF LEN(Z$)<1Z$=" "
400 PRINT CHR$(27,89,(35+L),87,27,75,27,89,32,98);: MEM : RETURN
410 PRINT CHR$(27,89,36,54);" ";A$;CHR$(27,75)
420 PRINT CHR$(27,89,37,54);" ";B$;CHR$(27,75)
430 PRINT CHR$(27,89,38,54);" ";C$;CHR$(27,75)
440 PRINT CHR$(27,89,39,54);" ";D$;CHR$(27,75)
450 PRINT CHR$(27,89,40,54);" ";E$;CHR$(27,75)
460 PRINT CHR$(27,89,41,54);" ";F$;CHR$(27,75)
470 PRINT CHR$(27,89,42,54);" ";G$;CHR$(27,75): RETURN
480 GOSUB 70: INPUT "IS THIS RECORD CORRECT (Y/N) "Z$: GOSUB 60
490 IF Z$="Y" RETURN
500 IF Z$<>"N" GOTO 480
510 RETURN
520 GOSUB 70: INPUT "WHAT LINE IS WRONG "L: GOSUB 60
530 IF L>7 GOTO 520
540 IF L<1 GOTO 520
550 PRINT CHR$(27,89,(35+L),54,14);" .............................";CHR$(15):
 GOSUB 360
```

```
560 IF L=1A$=Z$
570 IF L=2B$=Z$
580 IF L=3C$=Z$
590 IF L=4D$=Z$
600 IF L=5E$=Z$
610 IF L=6F$=Z$
620 IF L=7G$=Z$
630 PRINT CHR$(27,89,(35+L),54);" ";Z$;CHR$(27,75): RETURN
640 GOSUB 70: INPUT "ANOTHER RECORD (Y/N) "Z$: GOSUB 60
650 IF Z$="Y" RETURN
660 IF Z$="N" RETURN
670 IF Z$<>"N" GOTO 640
680 FOR L=1 TO 7: GOSUB 360
690 IF L=1A$=Z$
700 IF L=2B$=Z$
710 IF L=3C$=Z$
720 IF L=4D$=Z$
730 IF L=5E$=Z$
740 IF L=6F$=Z$
750 IF L=7G$=Z$
760 NEXT : RETURN
770 A$(A)=A$:B$(A)=B$:C$(A)=C$:D$(A)=D$:E$(A)=E$:F$(A)=F$:G$(A)=G$: RETURN
780 A$=A$(A):B$=B$(A):C$=C$(A):D$=D$(A):E$=E$(A):F$=F$(A):G$=G$(A): RETURN
790 A1=A1+1:X=A1:Z$=" ":A$(X)=Z$:B$(X)=Z$:C$(X)=Z$:D$(X)=Z$:E$(X)=Z$:F$(X)=Z$
800 G$(X)=Z$: RETURN
810 GOSUB 330: GOSUB 80: GOSUB 680
820 GOSUB 410
830 GOSUB 480
840 IF Z$="N" GOSUB 520: GOTO 830
850 GOSUB 770
860 A=A+1: GOSUB 640
870 IF Z$="Y" GOTO 810
880 A1=A -1: GOTO 170
890 GOSUB 70: PRINT "DATA SAVE MODE"
900 PRINT : PRINT : PRINT TAB(10);"DATE TAPE WAS LAST EDITED ";M$
910 PRINT : PRINT TAB(10);: INPUT "ENTER TODAY'S DATE "Z$: IF Z$<>" "M$=Z$
920 PRINT : PRINT TAB(10);
930 INPUT "SET TAPEDECK TO RECORD AND ENTER RETURN WHEN READY"Z$
940 DSAVE C: GOTO 170
950 GOSUB 70: PRINT "DATA LOAD MODE": PRINT : PRINT
960 PRINT TAB(10);: INPUT "SET TAPEDECK TO PLAY AND ENTER RETURN WHEN READY "Z$
970 DLOAD C
980 PRINT : PRINT : PRINT TAB(10);"DATE TAPE WAS LAST UPDATED ";M$
990 PRINT : PRINT : PRINT TAB(10);: INPUT "HIT RETURN TO CONTINUE "Z$
1000 GOTO 170
1010 GOSUB 70: INPUT "ADD RECORDS OR SEARCH FOR A RECORD (A/S) "Z$: GOSUB 60
1020 IF Z$="A"A=A1+1: GOTO 810
1030 IF Z$<>"S" GOTO 1010
1040 A=0: GOSUB 330
1050 GOSUB 70: INPUT "SEARCH WHAT LINE "L: GOSUB 60
1060 IF L<1 GOTO 1050
1070 IF L>7 GOTO 1050
1080 GOSUB 360
1090 T=LEN(Z$)
1100 A=A+1
1110 IF A>A1 GOSUB 70: INPUT "RECORD NOT FOUND, HIT RETURN TO CONTINUE "X$: GOSU
B 60: GOTO 1340
1120 IF L=1S$=MID$(A$(A),1,T)
1130 IF L=2S$=MID$(B$(A),1,T)
1140 IF L=3S$=MID$(C$(A),1,T)
```

```
1150 IF L=4S$=MID$(D$(A),1,T)
1160 IF L=5S$=MID$(E$(A),1,T)
1170 IF L=6S$=MID$(F$(A),1,T)
1180 IF L=7S$=MID$(G$(A),1,T)
1190 IF Z$<>S$ GOTO 1100
1200 GOSUB 780: GOSUB 410: GOSUB 80
1210 T$=Z$
1220 GOSUB 70: INPUT "IS THIS THE CORRECT RECORD "Z$: GOSUB 60
1230 IF Z$="N"Z$=T$: GOTO 1100
1240 IF Z$<>"Y" GOTO 1220
1250 GOSUB 70: INPUT "DELETE THIS RECORD (Y/N) "Z$: GOSUB 60
1260 IF Z$="N" GOTO 1320
1270 IF Z$<>"Y" GOTO 1250
1280 GOSUB 790:A1=A1-1
1290 FOR X=A TO A1:Y=X+1:A=X: GOSUB 80:X=A
1300 A$(X)=A$(Y):B$(X)=B$(Y):C$(X)=C$(Y):D$(X)=D$(Y):E$(X)=E$(Y):F$(X)=F$(Y)
1310 G$(X)=G$(Y): NEXT :A1=A1-1: GOTO 1340
1320 GOSUB 480
1330 IF Z$="N" GOSUB 520: GOSUB 770: GOTO 1320
1340 GOSUB 70: INPUT "SEARCH FOR ANOTHER RECORD (Y/N) "Z$: GOSUB 60
1350 IF Z$="Y" GOTO 1040
1360 IF Z$<>"N" GOTO 1340
1370 GOTO 170
1380 GOSUB 70: INPUT "PRINT LABELS OR PRINT ON PLAIN PAPER (L/P) "Z$: GOSUB 60
1390 IF Z$="L"T=1: GOTO 1420
1400 IF Z$<>"P" GOTO 1380
1410 T=0
1420 GOSUB 70: INPUT "HIT RETURN WHEN THE PRINTER IS READY TO PRINT "Z$: GOSUB 6
0
1430 POUT :N=0
1440 IF INUM(A1/2)=A1/2 GOTO 1460
1450 GOSUB 790:N=1
1460 L=0
1470 IF T=1 PRINT CHR$(27,69,27,71);
1480 FOR A=1 TO A1 STEP 2
1490 IF T=0 IF L=0 PRINT CHR$(10,10);TAB(20);M$;CHR$(10,10);
1500 PRINT TAB(2);A$(A);TAB(39);A$(A+1)
1510 PRINT TAB(2);B$(A);TAB(39);B$(A+1)
1520 PRINT TAB(2);C$(A);TAB(39);C$(A+1)
1530 PRINT TAB(2);D$(A);TAB(39);D$(A+1)
1540 IF T=1 GOTO 1590
1550 PRINT TAB(2);E$(A);TAB(39);E$(A+1)
1560 PRINT TAB(2);F$(A);TAB(39);F$(A+1)
1570 PRINT TAB(2);G$(A);TAB(39);G$(A+1)
1580 PRINT
1590 PRINT : PRINT
1600 IF T=0L=L+1: IF L=6 PRINT CHR$(12);:L=0
1610 NEXT
1620 IF T=1 PRINT CHR$(27,70,27,72);
1630 TOUT : IF N=1A1=A1-1
1640 GOTO 170
```

X

MEMORY CHECKSUM

-BY ROBERT CARR, 4691 FREEMAN ROAD, MIDDLEPORT, NY 14105 USA

THE FOLLOWING ROUTINE PROVIDES A CHECKSUM FOR ANY PLACE IN MEMORY. I WROTE IT AFTER DISCOVERING I HAD LOST SOME BITS DURING A TAPE LOAD, EVEN THOUGH I DID NOT GET A PARITY ERROR. IT IS ALSO USFUL FOR CHECKING EPROM'S FOR DATA DROP OUT.

FIRST, CHANGE C090H-C095H FROM 00 CF 0F CD CD CD TO 4B C3 BF 00 XX 0F. THIS CHANGES THE COMMAND TABLE OF THE MONITOR TO ACCEPT "K" AS THE COMMAND FOR CHECKSUM, WHERE XX IS THE ADDRESS OF YOUR STACK PAGE. THEN LOAD THE CODE FROM C3BFH-C3FFH AND YOUR ARE READY TO GO. ONE NOTE ON THE DISASSEMBLED LISTING. I AM USING CF00-CFFFH FOR MY STACK PAGE AND THE SCRT CALL'S TO CF15 SHOULD BE CHANGED TO XX15 WHERE XX IS YOUR STACK PAGE.

WRITING THIS PROGRAM DEMOSTRATED THE VALUE OF HAVING A STANDARD CLUB MONITOR IN THAT I WAS ALBE TO WRITE IT IN LESS THEN AN HOUR AND DEBUG IT IN 1/2 HOUR. USING THE BUILT IN ROUTINES REDUCED THE REQUIRED CODE OVER 80%. JUST BE VERY CAREFUL OF REGISTER USAGE SO YOU DO NOT DESTROY ANY REQUIRED BY YOUR PROGRAM.

```
C3BF  D4C141      SEP #C141
C3C2  F800        LDI #00
C3C4  BB          PHI RB
C3C5  AB          PLO RB
C3C6  B9          PHI R9
C3C7  A9          PLO R9
C3C8  ED          SEX RD
C3C9  2D          DEC RD
C3CA  8B          GLO RB
C3CB  F4          ADD
C3CC  AB          PLO RB
C3CD  9B          GHI RB
C3CE  7C00        ADCI#00
C3D0  BB          PHI RB
C3D1  89          GLO R9
C3D2  7C00        ADCI#00
C3D4  A9          PLO R9
C3D5  99          GHI R9
C3D6  7C00        ADCI#00
C3D8  B9          PHI R9
C3D9  D4C172      SEP #C172
C3DC  32E0        BZ  #E0
C3DE  30C8        BR  #C8
C3E0  D4C186      SEP #C186
C3E3  0D          LDN RD
C3E4  0A          LDN RA
C3E5  43      C   LDA R3
C3E6  48      H   LDA R8
C3E7  45      E   LDA R5
C3E8  43      C   LDA R3
C3E9  4B      K   LDA RB
C3EA  53      S   STR R3
C3EB  55      U   STR R5
C3EC  4D      M   LDA RD
C3ED  3D00        BN2 #00
C3EF  99          GHI R9
C3F0  D4CF15      SEP #CF15
C3F3  89          GLO R9
C3F4  D4CF15      SEP #CF15
C3F7  9B          GHI RB
C3F8  D4CF15      SEP #CF15
C3FB  8B          GLO RB
C3FC  D4CF15      SEP #CF15
C3FF  D5          SEP R5
```

UPGRADING TINY BASIC - NETRONICS VERSION

by M.E. Franklin, 690 Laurier Ave., Milton, Ont.

Many people got their first taste of programming a computer with one of the several versions of Tiny Basic on the market. Tom Pittman wrote an excellent version for the 1802, or rather versions, since Netronics, Quest and Pittman's own company, Itty Bitty Computers, marketed variations on the basic version, pardon the pun!

It has its shortcomings, but it is easy to use, particularily by novices or children, and is quite amenable to changes. Since I have a fanatical compulsion to tinker with things, I made it better.

I no longer use the 1861, nor Netronics monitor for the cassette I/O, therefore the code from 09D8 to 0E00 was deleted. The I/O vectors were changed to point to Symon - 0103 -C0 FE 03 (ASCII input) and 0106 - C0 FE 06 (ASCII output) - for some reason I had to change the C0 to a D4 to get the output driver to work for the VDU board.

Tiny is organized as follows:

initialization	0100 - 022C
TB Routines	022D - 0750
TB Interpreter	0766 - 09D7
ML Routines	0751 - 0765
" "	09E4 - 09F9
Cassette I/O	09FA - 0A5A
I/O	09D8 - 09E3
"	0A5A - 0DAF
"	00B6 - 00FF

In order to gain some working room, and to consolidate the machine language routines, I moved the TB Interpreter to 0900 - 0C00, and moved the FLG(ML routine from 09E4 to 076C, and the PEEK and POKE ML routines from 0118 and 0122 respectively to 0766 and 0768. Since the TB Interpreter code is self relative to its first address, as defined by a vector code at 011E, and the beginning of user RAM is simularily defined by a vector at 0120, only these two locations need be changed to allow Tiny to run properly after it has been relocated. Of course, you must change the TBIL code which calls the ML routines relocated above, but since these are now located following the main program, these will not need to be changed again if you decide to relocate the TB Interpreter again. We are now in a position to add additional statement or function commands to Tiny.

My first additions were: BYE - exit to monitor, and MEM - display available RAM, and to add Disk I/O commands compatible with my own SYDOS. Bye involved calling the coldstart entry to my monitor, which will only work if both Tiny and your monitor use the same Register conventions, duplicating the address twice, and executing a TBIL call, which does not return. Mem is alittle more complicated, requiring the routine to calculate the end point of the program, subtract that from the end of user RAM, store this value on the stack, and to print out the decimal value of the number on the stack. The code for the two above routines is as follows:

```
09D2  8A
      42  B
      59  Y
      C5  E
      E0      error if not CR
      0A      store two following bytes on stack
      FE
      00
```

```
09DA  0B   duplicate two top # on stack
      0B
      2E   call ml routine on stack

09DD  94
      4D   M
      45   E
      CD   M
      E0
      09      put variable 22 on top of stack
      22
      12      load variable 22 data onto stack
      09      repeat with variable 24
      24
      12
      19      subtract top four data bytes- leave
              answer on top of stack (availableRAM)
      24      print ASCII
      46   F
      52   R
      45   E
      C5   E
      22      tab 8 spaces
      20      print decimal value of top number on stack
      23      CR/LF
      1D      return for next command
```

As you can see from the above, the mystical intricacies of the TBIL are not insoliable, but rather easy, when approached by element. It is absolutly necessary to have a copy of Pittman's TB Experimenter's Kit manual to do this however. It is still available from Itty Bitty Computers PO Box 23189 San Jose Ca, USA 95153 for $10., along with several programming manuals, and a new game/instructional book - The First Book of Tiny Basic Programs. In the latter, Tom lists a speed up for the program line search function, which I have also incorporated into the working space at 080C -0846.

I will not detail the Disk commands or routines, since they specifically relate to my system.

My next endevour was to redefine the PLOT command to work with the ACE-VDU board. Tony Hill helped me set up the memory map addressing logic, for which I am grateful, and by some careful TBIL coding, and a ml routine, the old bit/alpha plot routine was born again, 64x32 bit matrix in either white or grey, and a 32x16 ASCII display, in either white or inverse.

I placed the PLOT TBIL code at 0A36, following the REM command, and the ml code at 0847 to 08A3. The TBIL code obtains the Y and X coordinates and the Alpha or Graphic character, then calls the ml routine to actually interface with the VDU driver. The PLOT command requires all three parms, unlike the origional TB version. Alpha is listed as ASCII hex codes 41-5F white dot is 00 and grey dot is 01. The graphic 4 mode is used, allowing 4 dots to be output at each Alpha position. The X/Y parms. determine which quadrant of the four is addressed. In this way, a 64x32 matrix can be

achieved while the ASCII format remains at 32x16. Obviously, ASCII letters utilize two adjacent address spaces, which must be remembered when calculating the X parm. in a program.

The code for the PLOT command is reproduced in the appendix.

Following implementation of the PLOT command, it became apparent that a Clear Screen CLS command was required to allow the PLOT to work at speed. This command simply calls a ml routine which outputs a home and clear screen 0C command to the VDU driver. The code is reproduced in the appendix.

One last comment which may help you before you dive into the program to implement your favourite command - and that purtains to its location in the TBIL. Statement commands are located first in the TBIL string, from LET to = (LET default), and are searched in that order, therefore frequently called statements should be located at the beginning if the TBIL string. Function commands are stored next, from math functions to the relational operators, and again searched in order. Note - do not move the math functions from address 018F (relative to the start of TBIL) since the function start address is calculated in the code to be at that address. If more code space is required for the statement commands, jump over the function area with a 3A70 command - jump to 0270.

TBIL commands may be any string of ASCII characters, with the last letter's msb set to 1. A short version of a name may also be incorporated, as in the PRINT command, by adding a skip if not ASCII match 8X code, as follows:

```
8A 52 45 D4 83 55 52 CE E0 15 1D xx
:   R  E  T  :  U  R  N
:              skip if not match to E0
skip to xx(next command) if not match
```

In the above, RETURN or RET would both match and therefore be executed, while REST or RE would not match and result in a skip to the next command.

As mentioned above, Tom Pittman has supported the 1802 version of TINY with a number of publications, including a book of programs,104 pages long, written in 1982. I spent my Christmas vacation keying in the Kingdom of Euphoria game and the Tiny Adventure game, some 6k and 24k respectively. Both are fun, and written with enough comments to help you modify the game, or apply the method to games you write yourself.

I did not write this article in a manner which allowed you to copy what I did with ease, if I did, you wouldn't have learned anything about Tiny's structure, nor achieved anything on your own. If you run into problems rewriting Tiny, then please write, listing your changes and commenting them so that I know what you intended to do, and I will try and help you.

Remember to keep an origional, unmodified copy of TINY separate from your working copy, so that you don't loose it completely if something goes wrong. Also back up each modification before you try if, for the same reason. I try to keep a paper copy of each modification until I know for certain that it works, then combine proven mods into a separate master working copy, separate from my origional version. In this way I always have a backup of known quality, and a copy of my current modification available if the code becomes destroyed by an unintented modification.

Appendix

TBIL code for PLOT

```
0A36  9E 50 4C 4F D4 0A 08 47 0A 00
0A40  20 32 5A 80 AC 0A 00 40 32 5A 01 02 0C 80 AC 31
0A50  8F E0 2E 0C 1D
```

Machine language code for PLOT

```
0847  8A FF 02 33 7A 88 FA 01 FE
0850  22 52 98 FA 01 F4 FF 00 32 6B FF 01 32 68 FF 01
0860  32 65 F8 41 C8 F8 42 C8 F8 44 C8 F8 48 BF 8A FF
0870  01 3A 77 9F FC 10 BF 12 30 7E 8A FA BF BF 98 BA
0880  F8 00 B8 F8 05 AF 88 F6 A8 88 FE A8 98 7E 88 2F
0890  8F 3A 89 22 88 52 9A F6 F4 A8 98 7C 00 FC E0 B8
08A0  9F 58 12 D5
```

TBIL code for CLS

```
0A55  8C 43 4C D3 E0 0A 08 A4 0B 0B 2E
0A60  0C 1D
```

Machine Language code for CLS

```
08A4  D4 C1 87 8C D5
```