Ipso Facto

ISSUE 32 DECEMBER, 1982

INDEX PAGE

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

Executive Corner	2
Editor's Corner	3
Members' Corner	4
Further Comments on Netronics Text Editor	6
FORTH Implementation Notes - III	8
Adding a String Editor to FORTH	10
16 Bit Integer Arithmetic for the 1802	12
A Simple "Cap Lock" Circuit for the 1802	21
ACE System Card Cage	22
Ace "Mini" Boot	23
The Euclidean Algorithm	24
The Netronics Smarterm - 80	28
"?" Search Subroutine for SYMON	34
Real Time Clock and Program	35
ACE Front Panel	37
Club Communique	40

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)	416-528-3222
	Fred Feaver (Boards)	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

A noticeable and disturbing trend is emerging in the relationship between our Members and the Newsletter. You will notice that if you review the index listing of the last six issues of Ipso Facto, that fewer and fewer articles are being written by the Membership at large. The greater portion are being written by the "home team". I no longer have a file of articles ready to print for future issues. This issue, and in fact, the last two, were rounded out only at the last minute.

Coupled with the lack of output by the Membership, are increased demands for particular types of software or answers to questions on "how to". Okay, so be it... but be prepared for a twenty page or even a four page Newsletter at some point. ACE is not a commercial/service Newsletter; it outputs what is input from the Membership as well as the Executive. Forced articles on solicited topics are often less satisfying than those written as a labour of love. Send us your programs in Tiny or "Big" Basic, Pilot, FORTH, Machine Language and your hardware applications - sharing with others is satisfying.

VDU BOARD UPDATE

The Membership response to our "limited time" offer for the VDU board was overwhelming. As a result, ACE has introduced a revised design offering several improved performance features over the original design, and a limited stock will be maintained for future orders.

ACE SYSTEM

Have you noticed? It now exists! ACE has completed an 1802 System - high performance, properly and fully decoded circuits, and inexpensive. A Backplane, CPU Card, Dynamic Memory, VDU, and Disk Boards give you a 64K System. The EPROM Board offers 16K of EPROM, or, for those who prefer, Static RAM Memory. The Front Panel offers a variety of user convenience features. Future projects contemplated include: a Modem, an A/D-D/A Board and an updated Disk Board for 5-1/4" Drives.

SOFTWARE

Membership response to "FORTH" has been very satisfying - over 70 copies distributed. FORTH offers excellent control and application features. ACE welcomes Membership articles on new words for the dictionary, application programs, etc. This month ACE is offering "Tiny Pilot" on cassette for distribution. Wayne Bowdish and Tony Hill have worked on the original "Petty Pilot" Kilobaud, July and December, 1979 and made a few minor improvements. This Pilot is functionally equivalent to Netronics Tiny Pilot as well. The program occupies 2K at 0000 and requires ASCII input and output routine from a system monitor. Pilot is a string matching program often used by schools for learning, spelling, and interactive program instructions. A "Line Editor" is also provided.

The COS DOS Operating System is in its final stages of completion. This program offers numerous high level program control and application features. While it is disk oriented, most of its features can be used with a cassette. Its availability and distribution format will be covered in February's issue of Ipso Facto.

FOR SALE:

M. Franklin, 690 Laurier Ave., Milton, Ontario. L9T 4R5 (416-878-0740)

1. ELF II 16K Static Memory Board (2114's) assembled & working. J. Ware Product. Fully socketed. $70.00 Ppd

2. Cybernex 64 char. by 16 line Video Display. Serial driver, parallel keyboard or micro input. 110 to 9600 baud rate.
Fully assembled and working. Fully socketed . $100.00 Ppd

Kevin Mast, 308 Jackson Ave., Defiance, Ohio. 43512 U.S.A. (419-782-6147)

NETRONICS ELF II Rev. C., cabinet, short course, 5 ea. 86 pin connectors - $95.00. Giant board - $20.00. 3 ea. 4K RAM Boards - $45.00 ea. Tiny Basic - short course - $10.00. Keyboard, VID Board, Cabinet - $100.00. Full Basic on cassette with math board - $60.00. Complete computer system - $400.00. All prices include shipping UPS.

ERRATA:

There were two "typos" in the "TBI BUBBLE SORT" on Page 15 of IF#30:

```
520  IF USR (S+20, A+I) > = USR (S+20, A+I - 1) GOTO 540

550  IF I < N GOTO 520
```

O&D BOARD UPDATE:

by T. Jones, Berliner Strasse 20, 6944 Hemsbach (FRG)

1. The new batch of "Q&D" Adapters (p.14, IF#30) is in. Sorry for those who were delayed. The new lower prices for quantity passed on to you are:

   $7.00 U.S. each tinned - $9.25 U.S. gold - both postpaid.

2. I would welcome suggestions for adapters, interfaces or other special circuit boards you need - the smaller the better, but even complicated projects will be of interest. What does your system need?

HELP!

F. Hanman, 10 Filosi Rd., East Lyne, Ct. USA 86333

I recently purchased a piece of equipment called a Microbuffer", which, is a 16K parallel buffer for my Epson Printer. I cannot get the thing to work. I even sent it back to the manufacturer and received a replacement which I can't get to work either. Would any Club Member who has the "Microbuffer" working, provide me with advice?

Wes Steiner
1204 2725 Melfa rd,
Vancouver,B.C. V6T 1N4

Dear ACE, Oct 14,1982

I was very dissappointed when I read of your plans to discontinue the search for a club standard Monitor.

I have been working on a number of software projects which are only waiting for the ACE monitor to make its appearence. I am not particularily excited about your development of a full-blown OS because I do not plan to purchase a disk drive for my system. I am very content to use two cassette players with a simplistic cassette tape OS. If I required the capabilities of a full OS then I would probably purchase an APPLE II or IBM PC.

I think you are making a big mistake by abandoning the standard monitor.

It doesn't have to be one of those 'every-thing-you-always-wanted to-do-with-hexidecimal-numbers' type of monitor. The only requirments would be:

1 A standard memory block location. eg. F800h.

2 A table of branch vectors in a standardized location where the user can insert the locations of his/her own I/O routines or patches.

3 And the ability to Inspect/Modify memory contents and to Jump to any memory location with P=X=0.

I don't think a monitor such as this would take too long to develop and it will work easily on any 1802 system and furthermore ACE would not have to try and write I/O routines to handle any possible TTY configuration because the individual users would supply their own. The important fact is that the branch vectors would be in the same place for any system.

I am very willing to help in any way I can in the development and distribution of a club monitor. First lets get this monitor distributed to the members and then everyone can start talking about the next steps: OS's, Editors, Compilers etc.

Wes Steiner

Editor's Note: I agree with you, but it seems that no agreement exists among our members as to where it should be located, what features it should offer, or how it should handle input/output of data, interrupts, etc. ACE has adopted a number of "conventions" to assist in software interchangeability. These include system level flags, ports, etc., and software - registers, SCRT and I/O vectors. ACE programs are written with 4 vectors (long branches) in the first page - ASCII characters IN, ASCII character OUT, BREAK and SYSTEM "CARRIAGE RETURN/LINE FEED". I personally feel the lack of software development for the 1802 can be attributed in part, to the lack of commonly accepted (and used) monitor and I/O.

Further Comments on Netronics Text Editor

- by Tom Pitman, Itty Bitty Computer

I just read Al Irwin's comments on "Netronics Text Editor Improvements". Perhaps I can fill in some of the historical details that explain why it is the way it is. I do not know one Mr. Larry Sandlin, so I cannot comment on his involvement with the RCA text editor, which is the basis of the Netronics version. I do know personally the original author of the editor, but I doubt anyone at RCA even remembers where it came from.

The editor was originally designed in 1974 to demonstrate the superiority of the 1802 (it was the 1801 then) over the 8080. This it did by being an exact copy (functionally) of the Intel text editor, which was a variant (as Al noticed) of Teco. The Intel version (and thus the copy) had several deficiencies, so once the superiority was established, or maybe in spite of it because RCA really did not know what they were doing, the editor was enhanced in-house by anonymous programmers (Sandlin?) who understood neither Teco nor the architecture of this program. Eventually the code stabilized at version 3.2, which ran with RCA's old disc non-operating system.

Netronics then acquired rights to distribute the editor and assembler, and engaged the services of Walter Petlevitch to adapt it to their cassette system. Walt was an early fan of my Tiny Basic, and as far as I know, he completed the adaptation. From his conversations with me on the phone at the time, I can only surmise that he thoroughly understood its operation (from disassembly and analysis, like Al), just as he did Tiny Basic. Since my connections with Netronics soured at that time (they quit responding to me), I lost track of what happened next. I did notice that Al's references to particular bytes correspond to my version (3.2), so I can assume that there is not much more in its history.

Yes, Teco was originally designed as a teletype-oriented editor, because CRTs did not exist in the 60's. Even in the mid-70's, the trusty ASR-33 was the cheapest terminal on the market by at least a factor of two or three (I paid $750 for my TTY when CRTs were running $2500). Escape and Rubout were the two control keys on the 33 that did not control the carriage, so they were adapted to the two control functions of an editor. Many teletypes had an "Altmode" key, which was a non-printing hex 7C, instead of the escape key, so Teco, and probably the Netronics editor also, accept any code from 7C-7E as if it were an escape. The code to check for that is in 0280-0285 (change the 32 at 0284 to 38 to inhibit this test).

Al says he detests using the escape as a command delimiter, "as most modern text editors use the return key." That is hardly the point. Teco is a character-oriented editor, not line-oriented like most modern editors. This gives it considerably more power, but also makes certain errors easier to make. Since returns are characters in the buffer just like any other character, Teco can make multiple-line edits, something generally impossible with line-oriented editors. For example (I do this all the time), I can build tables where certain parts of the code are the same, by typing in the differences of each entry on one line, separated by some unique character (such as "%"). Then I go through

the line with a repeated substitution, replacing the "%" with the duplicated part of the table, including all returns. Voila!. Instant tables. Or (try this on your line editor), I can type in text without concern over how long my lines are getting to be, or where the returns are. Now, to set the right margin so that all lines are about the same length, I start at the beginning of the buffer and execute a command like (where "r" means "return"):

```
B999<Sr$ $0L66CS $r$-T>$$
```

This command sequence will remove every existing return, then insert new returns so that each line ends with the next word after 66 characters. As for ending a command with a return being better for CRT terminals, that is baloney. It is annoying that it does not make use of the fact that on the CRT backspace and retype erases the old text, but once you get into the habit of ending your commands with "-9T9T$$" (which is fast on a CRT), that is no problem.

Al correctly observes that the original P command was patched out. "P" stands for "Punch", which in Teco means, "write out to the output file without deleting". It was in the original version, but was replaced in the RCA tinkering so they could add a Print command. In Teco, you did not print on the printer from the editor, you used PIP (a different program). RCA's Print operator is a hack. But I'm getting carried away. Lest anyone be misled, while I like Teco better than any line editor I've seen, it does not hold a candle to a good screen editor. Most of my work I now do on the UCSD system for just that reason.

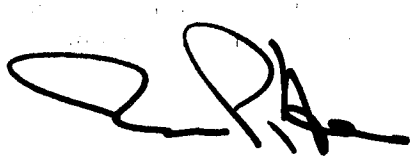
Two more tidbits: There is a bug in the Substitute command. If you hit break while it is searching, it will assume it found the target string and make the substitution wherever the pointer is at that instant (even though it did not match!). This is fixed by changing location 06CB from E5 to E6. Then it will tell you it can't find it, which is OK.

You can use the editor to justify your comments on an assembly source file (assuming RCA's syntax of two periods). Let's say you want the comments to start on column 24, unless the code on that line is longer. Use this command sequence:

```
B999<S ..$                    r..$-L23CF $KL-T>$$
```

The extra spaces before the return in the Substitute help to move the comment over to the right if it is too close. It is quite slow, especially if your program is large. When I use it I include an outer loop to Append and Write one block of 50 or 100 lines for every 20-30 comments justified in the inner loop. Be sure you start with the buffer over half full, so it will always find enough comments to justify (though in a thinly commented region, some may be justified several times). I usually put this run on just before going out for supper.

Tom Pittman
P.O. Box 6539
San Jose, CA 95150



FORTH IMPLEMENTATION NOTES - 3

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

This issue I have some more random FORTH notes and comments to pass along. Interest in 1802 FORTH seems to be picking up, as well over 50 club members now have copies. In fact, we are giving away free copies at club meetings to anyone who brings their own tape recorder and tape.

1)FORTH Reference Material

I have now read two FORTH reference books and feel I should pass along my observations.

The first book is called "Starting Forth" by Leo Brodie. This is one manual I would recommend to anyone using FORTH. It is written with both the novice and expert FORTH programmers in mind. The text is in clear ENGLISH, and contains many examples and illustrations. Writing a book which will be of interest to a broad range of programmer's is difficult at best, but Brodie seems to have pulled it off. Of particular interest to me was the more advanced examples at the end of the book, which included a FORTH file handling system.

The book does have one drawback. "Starting FORTH" is aimed at the FORTH INC. version of FORTH, which is slightly different than the fig version. In general, differences are noted and explained, but you may have some trouble getting the examples to run if you enter them exactly as shown in the text.

The other book I've looked at is called "Using FORTH" by FORTH Inc. This book is essentially a reference manual for the FORTH Inc. distribution of FORTH. I would not recommended it now that the Brodie book is available.

2)fig-FORTH Bugs - ?STACK

The fig-FORTH version of ?STACK does not check the upper or lower stack limit correctly. It's FORTH definition is:

```
: ?STACK  S0 @ DUP SP@ > 1 ?ERROR 256 + SP@ < 7 ?ERROR ;
```

Notice that this checks for a stack lower limit two bytes below the true bottom and for an upper limit that is in the return stack area in the standard distribution.

If your return stack and terminal buffer are on the same page as the computation stack the value of 256 used for the stack check should be reduced to about 120. Getting around the problem of the lower stack limit check will require a redefinition of the ?STACK word. The definition below will fit in the same space as the original version, but you will have to hand compile it and patch it into your own code if you want to replace the current definition.

```
: ?STACK SP@ S0 @ OVER OVER < 1 ?ERROR 120 + > 7 ?ERROR ;
```

Next issue I'll take a look at the problem with the way fig-FORTH handles the > and < words. The fig-FORTH version does funny things if you try to compare large but legal numbers!

3)Register Usage in fig-FORTH

One glaring omission from my FORTH articles so far has been an explanation of how the 1802 registers are used by FORTH.

First of all, R(7) and R(8) are used as temporary work registers within FORTH routines and are not assumed to have any value on entry to any FORTH word.

R(A) and R(B) are essentially used by FORTH as the FORTH program counters, keeping track of which words are being executed.

R(C) is the program counter for the inner interpreter, a little piece of code between 0091 and 009B that links all FORTH word execution together.

R(D) is the user variable pointer, pointing to the page allocated for variable storage.

R(9) is the computation stack pointer, which is the stack most commonly used to pass data with in FORTH. The R(9) stack looks like this:

	MEMORY ADDRESS	DATA		COMMENT
	n+2	--		- empty
	n+1	low byte		
grows ↑	n	high byte	<--R(9)	- top stack element
	n-1	low byte		
	n-2	high byte		- 2nd stack element
	n-3			

As shown in the picture R(9) point to the high byte of the top word on the stack. The stack is a "grow up" stack, as it move to higher memory addresses as it gets larger.

The R(2) stack works in the standard 1802 stack way. It is used to hold return addresses for the FORTH word linking mechanism as words are executed. The R(2) stack looks like:

	MEMORY ADDRESS	DATA		COMMENT
	n+4	high byte		-2nd stack element
	n+3	low byte		
grows ↓	n+2	high byte		-top stack element
	n+1	low byte		
	n	--	<--r(2)	-empty

4)Other Comments

I have had a number of calls from people using FORTH, and I know there are at least 50 people out there with copies of it. So how about sending in some articles. The article by Ken Mantei in this issue is the last in a series written by him about 15 months ago! If you want some article suggestions, how about some simple games that demonstrate the use of FORTH? Hope to hear from you soon

;s

1802 fig-FORTH "MATCH" And STRING EDITOR

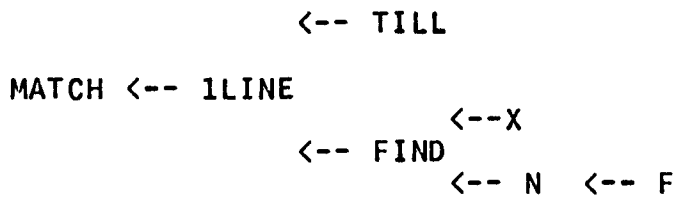
by- Ken Mantei, Cal State College, San Bernardino, Ca. USA 92407

Without an EDITOR, one can write, compile and test new FORTH words. However, if modifications are necesary, the whole definition must be typed in again--a chore, particularly since the original definition is not saved anywhere for reference. Adding an EDITOR allows modification of word definitions stored in RAM buffers.

Line editing words in fig-FORTH are written in FORTH, and are easily typed in from the fig-FORTH installation manual, if not already included in one's fig-FORTH. String editing provides the additional ability to modify portions of a line. The useful string editing words are: F, N, X, B and TILL (their functions are described in the installation manual). Implementation of string editing requires one key word, MATCH, for which neither an 1802 version (nor a high-level FORTH) definition is generally available. A tested 1802 code version of fig-FORTH MATCH is presented below.

The presented version of the MATCH word assumes the fig standard register assignments. R(2) is the return stack pointer, R(9) is the computation stack pointer, R(C) is the inner interpreter PC and R(7),R(8) are scratch registers. By simply modifying the MATCH code to reflect different register assignments, it should work in other 1802 FORTH implementations.

The dependence of the string editing words on MATCH can be symbolized as follows:



In addition, although none of these words is used in the definition of the B word, B makes no sense without prior exercise of F or N.

To add match to fig-FORTH, define the editor vocabulary if it does not already exist. Then type in the following:

```
EDITOR DEFINITIONS DECIMAL
CREATE MATCH SMUDGE 130 ALLOT
FORTH  DEFINTIONS  DECIMAL
LATEST 12 +ORIGIN !
HERE   28 +ORIGIN !
HERE   30 +ORIGIN !
HERE   FENCE !
' EDITOR 6 + 32 +ORIGIN ! (Do this only if EDITOR
                           is the latest
                           vocabulary to be added
                           to the system. )
```

Then exit FORTH, and using your systems monitor look at the end

of compiled FORTH definitions. The following should appear:

```
85 4D 41 54 43 C8   (MATCH name field)
xx xx               (Link to previous word)
yy yy               (Points to next byte)
130 bytes that may not appear "clean" but
    which are available for MATCH code
01 00               (where next word will go)
```

Copy the code for MATCH immediately following the yy yy bytes. There will be enough room to permit all branches to be changed to long branches if desired. Then save the new FORTH version for later use. Definitions for the remaining string editing functions are in FORTH and may be easily added to make fig-FORTH much more pleasant to work with.

Address															
XX00	99	B7	89	A7	27	07	19	E9	F4	73	27	07	74	73	29	29
XX10	27	27	27	07	52	22	F4	73	27	07	52	22	74	73	09	FC
XX20	00	30	27	29	09	FC	01	73	A8	09	7C	00	59	B8	19	19
XX30	19	19	49	B7	09	A7	29	29	29	29	49	19	F7	29	29	29
XX40	49	19	77	33	66	19	88	F7	29	98	77	33	23	07	E8	F3
XX50	E9	3A	23	19	19	19	19	19	17	18	87	F3	29	29	29	29
XX60	29	3A	45	F8	FF	C8	F8	00	A7	19	12	12	E2	88	F7	59
XX70	29	22	98	77	E9	73	87	73	59	19	19	12	DC			

Fig .1 1802 Code for fig-FORTH "MATCH" word. Since code will probably not start at a page boundary, underlined branches will have to be modified accordingly. If code crosses a page boundary, long branches need to be substituted for underlined short branches.

EDITOR'S NOTE: As mentioned in my article in the last issue, code definitions like MATCH can be typed in directly while running FORTH. For example, MATCH would be entered as follows:

```
EDITOR DEFINITIONS HEX
CREATE MATCH
99B7 , 89A7 , 2702 , 19E9 , F473 , 2707 , 7473 , 2929 ,
   (...continue with the rest of the code here...)
2922 , 9877 , E973 , 8773 , 5919 , 1912 , DC00 ,
SMUDGE
```

You can use:

```
HERE .
```

after you type CREATE MATCH to find out where the start address of the code will be.

...T. Hill

16-BIT INTEGER ARITHMETIC FOR THE 1802 W. BOWDISH

Many programs require basic arithmetic and logical operations on numbers which cannot be stored in a single byte. This article presents a basic set of routines which operate on 16-bit numbers. In a future article I will present a similar set of routines which operate on 32-bit numbers.

This set of routines operate on numbers which are stored in registers. Register A always contains an operand and always returns a value. For operations which require two operands such as ADDITION and SUBTRACTION,register B contains the second operand. The multiplication and division routines require temporary storage and so registers C and F(low) are used.

The following table lists the routines which are included in the package:

NAME	OPERATION	
SHL16	RA = RA (SHIFTED LEFT 1 PLACE)	
SHR16	RA = RA (SHIFTED RIGHT 1 PLACE)	
CMP16	RA = .NOT. RA	ONES COMPLIMENT OF RA
AND16	RA = RA .AND. RA	LOGICAL AND OF REGISTERS A AND B
OR16	RA = RA .OR. RB	INCLUSIVE OR
XOR16	RA = RA .XOR. RB	EXCLUSIVE OR
NEG16	RA = -RA	TWO'S COMPLIMENT OF REGISTER A
SUB16	RA = RA - RB	
ADD16	RA = RA + RB	
MUL16	RA = RA * RB	
DIV16	RA = RA / RB	

Most of the routines are quite straightforward if you understand the 1802 instruction set. However, the multiplication and division routines deserve some comments.

MULTIPLICATION

Binary multiplication is much simpler than decimal multiplication. The rules are as follows:

0 * 0 = 0	1 * 0 = 0
0 * 1 = 0	1 * 1 = 1

Using pencil and paper, a simple example might be 5*2 (I'll only use 4-bit numbers to keep the example simple);

```
                0010          2        <- MULTIPLICAND
                0101        * 5        <- MULTIPLIER

                ----        ---
                0010         10
partial         0000
products       0010
              0000

              -------
              0001010
```

Notice that multiplying 2 4-bit numbers could produce a result requiring up to 8 significant bits. When using the multiplication routine in the listing you must remember that only a 16-bit result can be returned. Ifthe product results in a number which requires more than 16-bits, the high order bits will be lost. I leave it up to the enterprising reader to produce a better version of MUL16 which returns a 32-bit result.

When performing the multiplication there are three points to note:

1. the product is the sum of a series of partial products,

2. ther is one partial product for each digit (or bit) of the multiplier. If a multiplier bit is 1, then the corresponding partial product is equal to the multiplicand. If the multiplier bit is 0 then the corresponding partial product is 0.

3. reading from bottom to top, the significant digits of each partial product are shifted one place to the right with respect to the partial product just below it.

Rather than generating all of the partial products and then summing them up, MUL16 will sum up the partial products as they are generated. The routine works as follows:

1. move the contents of register A to register C and zero register A. Register A will hold the sum of the partial products (ie. the resultant product when the routine exits).

2. set up a loop counter (RF.LO) and repeat the following steps once for each bit in the multiplicand (ie. 16 times).

3. shift register A left 1 place. This gives the same effect as if each partial product had been shifted one place to the right with respect to the one just below it.

4. shift register C (the multiplier) one place left. If the bit shifted out was a 1 then add register B to register A (remember the multiplication tables - if the bit was a 0 then the partial product is 0).

5. repeat steps 3 and 4 once for each bit in the multiplier (16 times)

DIVISION

Division is carried out using the rules of multiplication and subtraction. For example, dividing 212 (11010100) by 23 (00010111) gives:

```
                1001
            ------------
10111  )  11010100
           10111
           -------
           000111
            00000
           -------
           001110
            00000
           -------
           011100
            10111
           -------
           00101
```

Again there are a few points to note:

1. only significant bits of the divisor are used, leading zeros are eliminated. Thus, in effect, the divisor is shifted to the left until all leading zero bits have been shifted out.

2. the number of quotient digits (bits) is one more than the number of leading zeros that were shifted out of the divisor.

3. the numbers below the horizontal lines are partial remainders. The first partial remainder is the dividend while the last one is the remainder.

4. at each step in the division process, the divisor is compared to the current partial remainder. If the divisor is less than or equal to the partial remainder, it is subtracted from the partial remainder and a 1 is placed in the quotient. Otherwise a 0 is placed in the quotient and the partial remainder is unchanged.

The routine presented, DIV16, performs a 16-bit divide. The contents of register A are divided by the contents of register B. Internally, register A will hold the partial remainders and the quotient is formed in register C. At the end of the routine the contents of the registers are shuffled so that the quotient is returned in register A and the remainder is returned in register B.

DIV16 operates as follows:

1. check for an atempt to divide by zero. If so return immediately with DF set. Note that if the division is successful then, on return, DF will be reset.

2. shift the divisor (RB) left until the most significant bit is a 1. Register F (lo) will contain a count of the number of zeros shifted out. RF.LO is then incremented to reflect the number of digits in the quotient.

3. Clear register C which will contain the quotient. The following steps will be repeated once for each digit in the quotient

4. shift register C (the quotient) left 1 place to make room for the next quotient bit to be inserted

5. if the divisor (RB) is less than or equal to the partial remainder (RA) subtract RB from RA and insert a 1 bit into the quotient (RC)

6. repeat steps 4 and 5 once for each digit in the quotient. When done move the quotient and remainder in to the apprpriate registers and return.

Well thats it for 16-bit arithmetic. I hope that these routines will answer some questions for the programmer new to 1802 arithmetic.

```
                                .ORG    #0100
                        ;
                        ;                       M A T H 1 6
                        ;
                        ;               A 16-BIT MATH PACKAGE FOR THE 1802
                        ;
                        ;   THIS COLLECTION OF ROUTINES PERFORMS BASIC ARITHMETICAL AND
                        ;   LOGICAL OPERATIONS ON 16-BIT NUMBERS. THE VALUES ARE PASSED
                        ;   TO THE ROUTINES IN REGISTERS, RA IS ALWAYS A DESTINATION OR
                        ;   ACCUMULATOR REGISTER ( CALLED THE AC ) AND RB IS USED AS AN
                        ;   OPERAND REGISTER ( CALLED THE OP ).  FOR MULTIPLICATION AND
                        ;   DIVISION A TEMPORARY REGISTER RC ( CALLED THE MQ ) IS USED.
                        ;
                        ;   S H L 1 6 - LOGICAL SHIFT LEFT
                        ;
                        ;       RA = RA ( LEFT SHIFTED 1 BIT POSITION )
                        ;
0100                    SHL16:
0100  8A                        GLO     RA      ;\
0101  FE                        SHL             ; SHIFT LOW BYTE LEFT
0102  AA                        PLO     RA      ;/
0103  9A                        GHI     RA      ;\
0104  7E                        SHLC            ; SHIFT HIGH BYTE PLUS CARRY
0105  BA                        PHI     RA      ;/
0106  D5                        +RETRN
                        ;
                        ;   S H R 1 6 - LOGICAL SHIFT RIGHT
                        ;
                        ;       RA = RA ( RIGHT SHIFTED 1 BIT POSITION )
                        ;
0107                    SHR16:
0107  9A                        GHI     RA      ;\
0108  F6                        SHR             ; SHIFT HIGH BYTE
0109  BA                        PHI     RA      ;/
010A  8A                        GLO     RA      ;\
010B  76                        SHRC            ; SHIFT LOW BYTE PLUS CARRY
010C  AA                        PLO     RA      ;/
010D  D5                        +RETRN
                        ;
                        ;   C M P 1 6 - ONE'S COMPLIMENT OF RA
                        ;
                        ;       RA = .NOT. RA
                        ;
010E                    CMP16:
010E  9A                        GHI     RA      ;\
010F  FB FF                     XRI     #FF     ; COMPLIMENT HIGH BYTE
0111  BA                        PHI     RA      ;/
0112  8A                        GLO     RA      ;\
0113  FB FF                     XRI     #FF     ; COMPLIMENT LOW BYTE PLUS CARRY
0115  AA                        PLO     RA      ;/
0116  D5                        +RETRN
                                .SLW
```

```
                        ;
                        ;    A N D 1 6 - LOGICAL AND
                        ;
                        ;       RA = RA .AND. RB
                        ;
0117                    AND16:
0117  8B                       GLO     RB
0118  52                       STR     R2
0119  8A                       GLO     RA
011A  F2                       AND
011B  AA                       PLO     RA
011C  9B                       GHI     RB
011D  52                       STR     R2
011E  9A                       GHI     RA
011F  F2                       AND
0120  BA                       PHI     RA
0121  D5                       +RETRN
                        ;
                        ;    O R 1 6 - LOGICAL OR
                        ;
                        ;       RA = RA .OR. RB
                        ;
0122                    OR16:
0122  8B                       GLO     RB
0123  52                       STR     R2
0124  8A                       GLO     RA
0125  F1                       OR
0126  AA                       PLO     RA
0127  9B                       GHI     RB
0128  52                       STR     R2
0129  9A                       GHI     RA
012A  F1                       OR
012B  BA                       PHI     RA
012C  D5                       +RETRN
                        ;
                        ;    X O R 1 6 - EXCLUSIVE OR
                        ;
                        ;       RA = RA .XOR. RB
                        ;
012D                    XOR16:
012D  8B                       GLO     RB
012E  52                       STR     R2
012F  8A                       GLO     RA
0130  F3                       XOR
0131  AA                       PLO     RA
0132  9B                       GHI     RB
0133  52                       STR     R2
0134  9A                       GHI     RA
0135  F3                       XOR
0136  BA                       PHI     RA
0137  D5                       +RETRN
                               .SLW
```

```
                        ;
                        ;    N E G 1 6 - ARITHMETIC NEGATE ( TWOS COMPLIMENT )
                        ;
                        ;       RA = -RA
                        ;
                        ;    THE NEGATION IS DONE BY SUBTRACTING THE CONTENTS OF
                        ;    REGISTER-A FROM 0. ( IE. RA = 0 - RA )
                        ;
0138                    NEG16:
0138  8A                      GLO     RA      ;\
0139  FD 00                   SDI     0       ; NEGATE LOW BYTE
013B  AA                      PLO     RA      ;/
013C  9A                      GHI     RA      ;\
013D  7D 00                   SDBI    0       ; NEGATE HIGH BYTE
013F  BA                      PHI     RA      ;/
0140  D5                      +RETRN
                        ;
                        ;    S U B 1 6 - SUBTRACT ROUTINE
                        ;
                        ;       RA = RA - RB
                        ;
0141                    SUB16:
0141  8A                      GLO     RA      ;\
0142  52                      STR     R2      ; \
0143  8B                      GLO     RB      ;  RA.LO = RA.LO - RB.LO
0144  F5                      SD              ; /
0145  AA                      PLO     RA      ;/
0146  9A                      GHI     RA      ;\
0147  52                      STR     R2      ; \
0148  9B                      GHI     RB      ;  RA.HI = RA.HI - RB.HI
0149  75                      SDB             ;
014A  BA                      PHI     RA      ;/
014B  D5                      +RETRN
                        ;
                        ;    A D D 1 6 - ADDITION ROUTINE
                        ;
014C                    ADD16:
014C  8A                      GLO     RA
014D  52                      STR     R2
014E  8B                      GLO     RB
014F  F4                      ADD
0150  AA                      PLO     RA
0151  9A                      GHI     RA
0152  52                      STR     R2
0153  9B                      GHI     RB
0154  74                      ADC
0155  BA                      PHI     RA
0156  D5                      +RETRN
                              .SLW
```

```
                        ;
                        ;    M U L 1 6 - MULTIPLICATION ROUTINE
                        ;
                        ;       RA = RA * RB
                        ;
0157                    MUL16:
0157  9A                        GHI     RA          ;\
0158  BC                        PHI     RC          ; COPY AC ( RA ) TO MULTIPLIER
0159  8A                        GLO     RA          ; QUOTIENT ( MQ ) REGISTER
015A  AC                        PLO     RC          ;/
015B  F8 00                     LDI     0           ;\
015D  BA                        PHI     RA          ; ZERO AC
015E  AA                        PLO     RA          ;/
015F  F8 10                     LDI     16          ; RF IS A LOOP COUNTER TO CONTROL
0161  AF                        PLO     RF          ;  LOOP FOR EACH BIT IN AC
0162                    MUL010:
0162  D4 01 00                 +CALL   SHL16       ; SHIFT AC LEFT 1 PLACE
0165  8C                        GLO     RC          ;\
0166  FE                        SHL                 ; \
0167  AC                        PLO     RC          ;  SHIFT MQ LEFT 1 PLACE
0168  9C                        GHI     RC          ;
0169  7E                        SHLC                ; /
016A  BC                        PHI     RC          ;/
016B  3B 70                     BNF     MUL020      ; BIT SHIFTED OUT = 0
016D  D4 01 4C                  +CALL   ADD16       ; BIT SHIFTED OUT = 1 SO ADD OF TO AC
0170                    MUL020:
0170  2F                        DEC     RF          ;\
0171  8F                        GLO     RF          ; LOOP FOR ALL 16 BITS
0172  3A 62                     BNZ     MUL010      ;/
0174  D5                        +RETRN
                        ;
                        ;    D I V 1 6 - DIVISION ROUTINE
                        ;
                        ;       RB = RA / RB  ( REMAINDER RETURNED IN RB )
                        ;
0175                    DIV16:
0175  9B                        GHI     RB
0176  3A 7B                     BNZ     DIV010
0178  8B                        GLO     RB
0179  32 BA                     BZ      DIV070
017B                    DIV010:
017B  F8 00                     LDI     0           ; RF.LO WILL BE USED TO COUNT THE
017D  AF                        PLO     RF          ;  NUMBER OF DIGITS IN THE QUOTIENT
                                .SLW
```

```
 1 017E                 DIV020:
 2 017E  9B                     GHI     RB      ;\
 3 017F  FA 80                  ANI     #80     ; \
 4 0181  3A 8C                  BNZ     DIV030  ;  \
 5 0183  8B                     GLO     RB      ;   \
 6 0184  FE                     SHL             ;    SHIFT OP LEFT UNTIL LEFTMOST SIGNIFICANT
 7 0185  AB                     PLO     RB      ;    BIT IS A 1.  RF.LO WILL CONTAIN A COUNT
 8 0186  9B                     GHI     RB      ;    OF NUMBER OF 0-BITS SHIFTED OUT
 9 0187  7E                     SHLC            ;   /
10 0188  BB                     PHI     RB      ;  /
11 0189  1F                     INC     RF      ; /
12 018A  30 7E                 BR      DIV020  ;/
13 018C                 DIV030:
14 018C  1F                     INC     RF      ; INC. COUNT TO GIVE NUM. OF BITS IN QUOTIENT
15 018D  F8 00                  LDI     0       ;\
16 018F  BC                     PHI     RC      ; CLEAR MQ
17 0190  AC                     PLO     RC      ;/
18 0191                 DIV040:
19 0191  8C                     GLO     RC      ;\
20 0192  FE                     SHL             ; \
21 0193  AC                     PLO     RC      ;  SHIFT MQ LEFT 1 PLACE
22 0194  9C                     GHI     RC      ;
23 0195  7E                     SHLC            ; /
24 0196  BC                     PHI     RC      ;/
25 0197  D4 01 41               +CALL   SUB16   ; SUBTRACT OP FROM AC
26 019A  33 A1                  BDF     DIV050  ;\
27 019C  D4 01 4C               +CALL   ADD16   ; IF CARRY THEN UNDO THE SUBTRACT
28 019F  30 A5                 BR      DIV060  ;/
29 01A1                 DIV050:
30 01A1  8C                     GLO     RC      ;\
31 01A2  F9 01                  ORI     #01     ; NO CARRY SO OR 1 INTO LEAST SIGNIFICANT BIT
32 01A4  AC                     PLO     RC      ;/
33 01A5                 DIV060:
34 01A5  9B                     GHI     RB      ;\
35 01A6  F6                     SHR             ; \
36 01A7  BB                     PHI     RB      ;  SHIFT OP RIGHT 1 PLACE
37 01A8  8B                     GLO     RB      ;
38 01A9  76                     SHRC            ; /
39 01AA  AB                     PLO     RB      ;/
40 01AB  2F                     DEC     RF      ;\
41 01AC  8F                     GLO     RF      ; LOOP IF ALL DIGITS IN QUOTIENT NOT PROCESSED
42 01AD  3A 91                  BNZ     DIV040  ;/
43 01AF  8A                     GLO     RA      ;\
44 01B0  AB                     PLO     RB      ; PUT REMAINDER INTO RB
45 01B1  9A                     GHI     RA      ;
46 01B2  BB                     PHI     RB      ;/
47 01B3  8C                     GLO     RC      ;\
48 01B4  AA                     PLO     RA      ; PUT QUOTIENT INTO RA
49 01B5  9C                     GHI     RC      ;
50 01B6  BA                     PHI     RA      ;/
51 01B7  FC 00                  ADI     0       ; RESET DF
52 01B9  D5                     +RETRN
53 01BA  FF 00          DIV070: SMI     0       ; SET DF IF ERROR DETECTED
54 01BC  D5                     +RETRN
```

A SIMPLE 'CAPS-LOCK' CIRCUIT

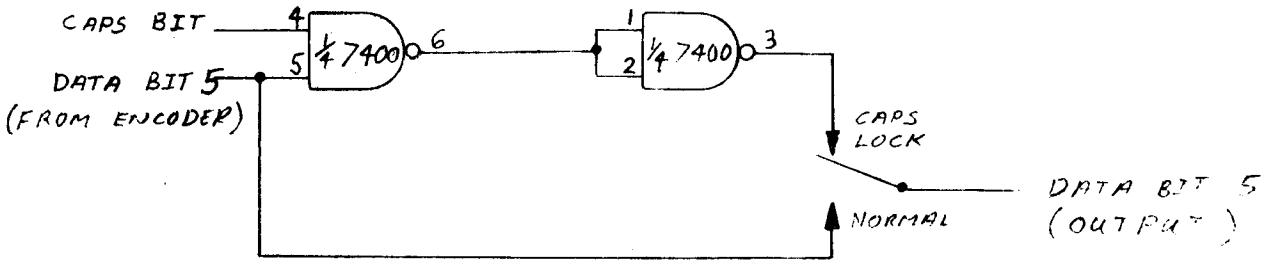
W. Bowdish

Some time ago I purchased a used ASCII keyboard for my system. The keyboard has a single chip keyboard encoder which is capable of generating the full 128 ASCII character set. Unfortunately it is a typewriter style keyboard. The unshifted output consists of lower case alphabetic characters, digits and some special characters. The shift key causes the upper case characters to be generated. This setup is fine for many applications but most monitors and languages only accept upper case letters. What to do?

When the keyboard was purchased (several people got them at about the same time) Frank Ditomaso spent many hours determining how the keyboards worked (you don't get much documentation for $20). Frank found that the encoder chip generated a rather interesting bit (I'll call it the CAPS bit) along with the usual data and parity bits. The following table shows how this bit is related to the ASCII codes.

ASCII code (hex)	CAPS bit	ASCII data bit 5
00 - 1F	0	0
20 - 3F	1	1
40 - 5F	0	0
60 - 7F	0 for lower case (61-7F) 1 for all other characters	1

In order to convert lower case to upper case, all that has to be done is to logically AND data bit 5 and the CAPS bit and use the result as data bit 5 of the character. The following schematic shows a circuit which will do this. I had a 7400 in the junk box so that is what the circuit is based on. The switch was mounted on the keyboard case and labeled appropriately.



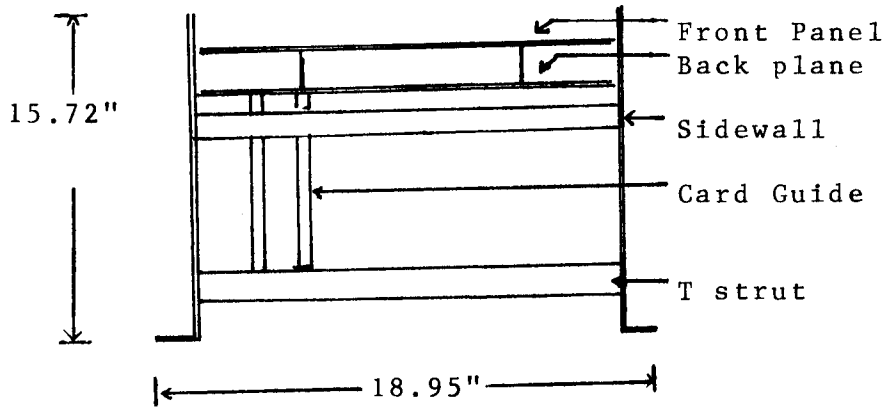
- by R. Poore, 2636 Fulbourne Drive., Cincinatti, Ohio, U.S.A. 45231

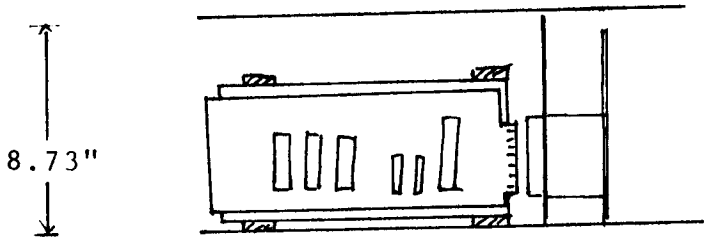
Now that a whole line of boards is available for the 1802 from ACE, a home for them is needed. Vector Electronic Company (12460 Gladstone Avenue, Sylmar, CA 91342) has the parts necessary to build a rack mount card cage for ACE boards.

Parts Required: (Prices from 11/9/81 Price List)

2	SW - 87 - P156	CAGE SIDE WALL, 8.73 X 15.72"	$18.74
4	TS169-4	T-STRUT, 16.85" W.	$13.96
3	BR20 - 10HP	SCREWMOUNT PLASTIC CARDGUIDE 8.52" L - PACK OF 12	$21.27
			$53.97

Approximately 4" are free in the back for mounting a power supply, etc. Some method will have to be found to mount the backplane to the cage.



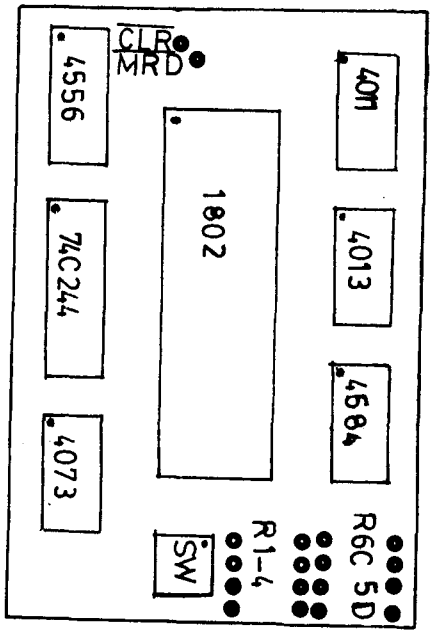


ACE Mini-Boot
-by K.Bevis, B. Murphy and M. Franklin

Unlike the camel designed by committee, this circuit works. A BOOT is a devise which resets the micro chip upon power-up and causes the micro to address a predetermined location through the use of hardware instead of software. It is the kind of a circuit that is a real convenience!

ACE developed a BOOT for the ACE CPU board, and as a benefit to other members, developed a 2"x3" board as an add-on upgrade of existing 1802 boards, such as the ELF**II**, TEC or Quest Super ELF.

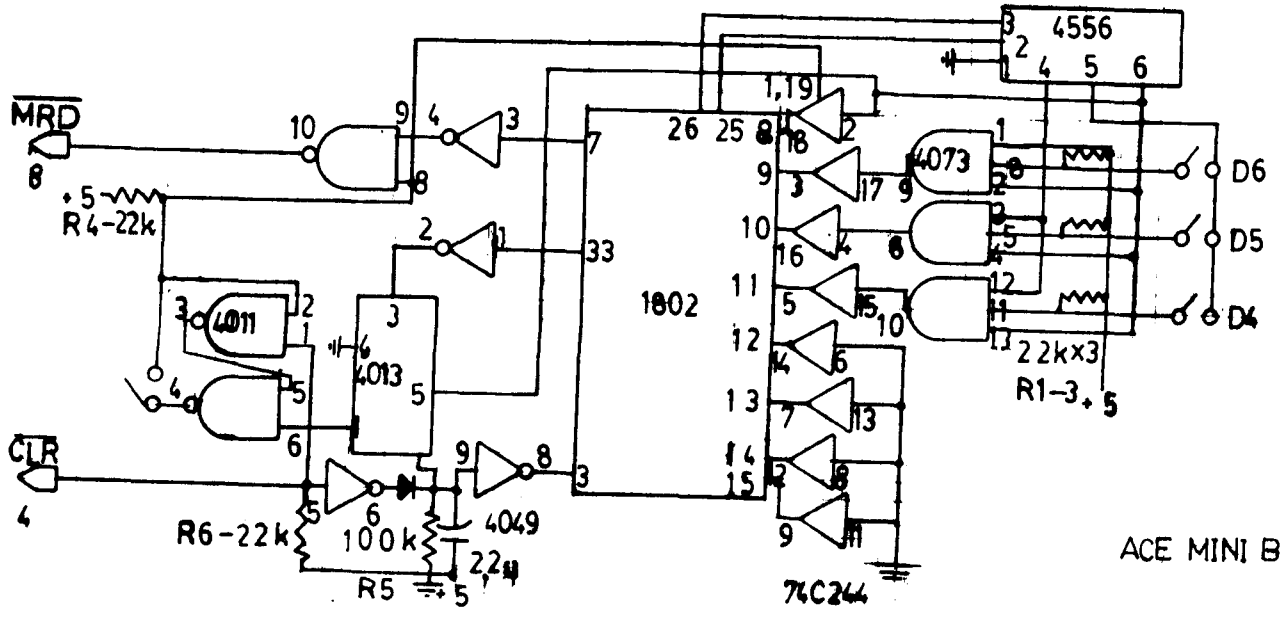
The origional CPU and socket are removed from the motherboard. A wire wrap socket is soldered to the mini-boot board and this, in turn, is soldered to the motherboard. The length of the wire wrap pins ensures that the board can be placed over near-by ICs.



$\overline{MRD}$ and $\overline{CLR}$ traces are cut between the 1802 and the buss, or other circuit, and these buss traces are now connected to the two labeled holes on the board. The user may replace the RESET/RUN switch with a momentary or spring loaded switch for convenience.

Upon power-up, the R-C delay pulses CLR to reset the 1802. The NAND gate and flip-flop disable $\overline{MRD}$, preventing a memory read. Addresses A0 and A1 cycle the 4556, producing three low pulses on the 3 input AND gate, simulating a long jump command. A0002 resets the flip-flop allowing $\overline{MRD}$ to cycle normally. The 4 position switch selects the boot address of D4 to D6, (D7 is always high), and position 4 disables the boot for normal DMAIN operation. The ACE-BOOT can select a 4k boundry in the range of 8000H to F000H.

The board is available from ACE for $6.00.



ACE MINI BOOT

THE EUCLIDEAN ALGORITHM

by - Bob Smith, 1 Cranleigh Court, Islington, Ontario, Canada, M9A 3Y2

The purpose of this article is to describe a programme, written in Tiny Basic, of an efficient algorithm for computing the greatest common divisor of any two numbers A and B, not both zero. The algorithm is due to Euclid and uses only integer arithmetic. Because of Tiny Basic's size restrictions, the numbers A and B must lie between -32,767 and 32,767. In full Basic, the programme can be somewhat shortened and, of course, the range of A and B will automatically be increased; the details can be supplied by the interested reader.

BACKGROUND TO THE EUCLIDEAN ALGORITHM

The greatest common divisor (GCD) of A and B is the largest positive integer D which divides both A and B. For example, if A = 15 and B = 21, then D = 3. It is often convenient to write GCD(A,B) for D. Note that if A = B = 0, every positive integer is a common divisor of A and B and so the GCD in this case doesn't make much sense; for this reason, we ignore this case. Aside from being able to spot the GCD in some cases, it is seldom easy to spot the GCD in general. For example, what is the GCD of 12,614 and 30,634? Euclid's algorithm provides a quick answer with the 1802. The idea behind the algorithm is the following.

First of all, the division algorithm says the following. If X and Y are any two integers with $X \neq 0$, then we can always find unique integers Q and R such that

$$Y = QX + R \quad \text{with} \quad 0 \leq R < |X|.$$

Here, $|X|$ is the absolute value of X and QX denotes the product of Q and X, which in Basic would be written as Q*X. It is particularly easy to determine the quotient Q and the remainder R with Tiny Basic. They are just:

$$Q = Y/X \quad \text{and} \quad R = Y - Q*X,$$

provided that X and Y lie between -32,767 and 32,767.

To determine the GCD of A and B, let $X_0 = A$ and $X_1 = B$. By the division algorithm, we can find unique integers X_2 and Q_1 such that $X_0 = Q_1X_1 + X_2$, where $0 \leq X_2 < X_1$. If $X_2 \neq 0$, we may again apply the division algorithm to find unique integers X_3 and Q_2 such that $X_1 = Q_2X_2 + X_3$ with $0 \leq X_3 < X_2$. By repeating this process, it will eventually come to a stop since the remainders

get smaller and smaller and must eventually reach zero. So, we obtain a unique set of integers $X_2, X_3, \ldots, X_n$ and $Q_1, Q_2, \ldots, Q_n$ such that

$$\begin{aligned} X_0 &= Q_1 X_1 + X_2 \\ X_1 &= Q_2 X_2 + X_3 \\ &\cdots\cdots\cdots\cdots\cdots \\ X_i &= Q_{i+1} X_{i+1} + X_{i+2} \\ &\cdots\cdots\cdots\cdots\cdots \\ X_{n-2} &= Q_{n-1} X_{n-1} + X_n \\ X_{n-1} &= Q_n X_n + 0, \end{aligned} \tag{1}$$

where $0 = X_{n+1} < X_n < X_{n-1} < \ldots < X_2 < X_1 < |B|$. Note the nice diagonal pattern the X's satisfy in (1). Here, n is the number of times we must apply the division algorithm before the remainder reaches zero. The last non-zero remainder X_n is the GCD of A and B. The explaination of this is rather complicated and so you will just have to accept that it is true. In any event, this provides a simple way of computing the GCD of A and B — just keep applying the division algorithm over and over until you hit the last non-zero remainder. This is Euclid's algorithm for finding GCD(A,B). With a bit of effort, you can programme this on your 1802.

To illustrate how this algorithm works, let's go gack to the original example with A = 15 and B = 21 and apply the algorithm to it. We immediately obtain:

$$\begin{aligned} 15 &= 0*21 + 15 \\ 21 &= 1*15 + 6 \\ 15 &= 2*6 + 3 \\ 6 &= 2*3 + 0 \end{aligned} \tag{2}$$

and so $Q_1 = 0$, $Q_2 = 1$, $Q_3 = 2$, $Q_4 = 2$ and $X_0 = 15$, $X_1 = 21$, $X_2 = 15$, $X_3 = 6$, $X_4 = 3$, $X_5 = 0$. Thus, GCD(15,21) = $X_4 = 3$, as we previously found by inspection. If we examine the calculations in (2) more closely, we observe that the second and third calculations imply that $3 = 15 - 2*6 = 15 - 2*(21 - 15) = 3*15 - 2*21$, that is, the GCD of 15 and 21 can be expressed in terms of the original two numbers. This is no accident: it always works this way. In fact, if you start with the ith equation in (1) (for any i = 0,1,2,...,n-1), and work backwards, you will find that X_i can always be expressed in terms of X_0 and X_1, that is,

$$X_i = F_i X_0 + H_i X_1 \qquad (3)$$

for some numbers F_i and H_i which change with your choice of i. In particular, from the above discussion, we then have

$$GCD(A,B) = X_n = F_n X_0 + H_n X_1 = F_n A + H_n B. \qquad (4)$$

The question now is: How can we find these numbers F_n and H_n? If you substitute (3) into the ith equation in (1), you will find that the conditions we want the F's and H's in (3) to satisfy are

$$F_{i+2} = F_i - Q_{i+1}F_{i+1} \quad \text{and} \quad H_{i+1} = H_i - Q_{i+1}H_{i+1} \qquad (5)$$

for i = 0,1,...,n-2. For i = 0 and 1, equation (3) becomes $X_0 = F_0X_0 + H_0X_1$ and $X_1 = F_1X_0 + H_1X_1$. These two equations are satisfied by taking

$$F_0 = 1,\ F_1 = 0,\ H_0 = 0 \text{ and } H_1 = 1.$$

These are the initial conditions which must be supplied to solve for F_n and G_n by means of the equations in (5). These initial conditions occur in the programme at line numbers 130 through 160.

If you have enough patience, you can solve these equations by hand if A and B are not too large. This procedure of course requires good book keeping habits and plenty of paper! However, hand calculations are a thing of the past for such repetitive calculations when you can call upon your 1802 to do the job for you. I suggest that before you programme your CPU for this task, you should do one or two by hand to understand how the programme works, and of course to better appreciate your 1802's computing power. For example, by hand, find

GCD(1802,1806) and GCD(1802,1805)

and express both in terms of the type of result given in (4). After you've done this once (or twice, or maybe three times if your eraser has worn out!), you will discover that the 1802 will do these calculations in a second or two if you're using Palo Alto Tiny Basic with a 3.579545 MHz crystal driving the CPU (I'm using the ELF II); if you're using Netronic's Tiny Basic, the calculations will take two or three times as long.

THE PROGRAMMED VERSION OF THE EUCLIDEAN ALGORITHM

After you have typed in my Tiny Basic implementation of these two algorithms

and set the CPU to RUN the programme, you will find immediately a data request for the values of A and B. Once these have been provided, the print out gives the value of the GCD(A,B) followed by the values of X and Y satisfying GCD(A,B) = AX + BY given by (4), where $X = F_n$ and $Y = H_n$. If we were to stick to positive integers A and B only, the number of steps in the programme given below could be roughly reduced by half. However, I wrote it up to deal with any integers A and B satisfying $|A| < 32{,}767$ and $|B| < 32{,}767$, the largest size integers that Tiny can directly handle. Admittedly, no one in his right mind would care about the GCD of -1802 and -1806, but here it is anyway.

```
10   PLOT 12
20   PRINT
30   PRINT"The GCD of A and B can be written as AX + BY."
40   INPUTA,B
50   LET U = A
60   LET V = B
70   IF A = Ø IF B = Ø GOTO 330
80   IF A<>Ø IF B<>Ø GOTO 120
90   IF A<>Ø GOTO 120
100  LET A = B
110  LET B = Ø
120  LET T = 1
130  LET E = 1
140  LET F = Ø
150  LET G = Ø
160  LET H = 1
170  LET Q = B/A
180  LET R = B - Q*A
190  IF R = Ø GOTO290
200  LET M = F
210  LET N = H
220  LET F = E - Q*F
230  LET E = M
240  LET N = G - Q*H
250  LET G = N
260  LET B = A
270  LET A = R
280  GOTO 170
290  IF A < Ø LET T = -1
300  PR"GCD(";U;",";V;") = ";T*A, "","X=";
305  IF U<>Ø PR T*H, "Y=";T*F
310  IF U = Ø PR T*F, "Y=";T*H
320  GOTO 40
330  PR"The GCD of Ø and Ø is not defined."
340  GOTO 40
```

REMARKS

1) The execution time can be somewhat decreased by deleting the "LET" statements from the programme.

2) By adding the new programme line

you will get a complete print out of the operation of the division algorithm that produces the GCD of A and B. I should add that it will only work correctly if A and B are positive integers.

THE NETRONICS SMARTERM-80

M. Smith

This article is the first in a series on the various new terminals and Video Boards which are available on the market today. If you are interested in purchasing a video board for your system, these articles will give you a lot of valuable information concerning your future purchase.

The Netronics Smart Video Board has quite a few nice features for its $369.95 (assembled board plus keyboard, cabinet and power supply; unassembled version is 299.95 dollars, American) price tag:

- 8085 microprosessor
- crystal speeds are 7, 10 or 18 Mhz and 6.144 Mhz
- 2K of RAM
- up to 4K of ROM
- 80 by 24 or 40 by 16 screen format
- switch selectable serial baud rate
- 128 printable ASCII characters
- full cursor control
- absolute cursor addressing
- graphics
- visual attributes: underline, blink, reverse video, and half intensity
- printer output under program control

The printed circuit board for the Smarterm-80 is slightly less then the usual Netronics quality but the layout is of the same good quality. The assembly instructions, schematic and user manual are also of good quality.

A few things, though, are missing from the user manual—namely high baud rate I/O patches for Tiny Basic or other languages. To take full advantage of the high speed features of this board, having a high baud rate is almost a must.

ASSEMBLY:

There are no real problems with getting the board operational, or with hooking it up to the 1802 and keyboard. A standard 14 pin cable connects the video board to the keyboard and 3 wires—one ground and the other two are for data—connect it to your Giant Board. The user may also wish to take advantage of the many switch selectable options, so numerous switches could be run to the back of the cabinet if needed.

These options, and others, will be discussed later.

OPERATION:

This board has the standard ASCII character set and cursor commands with numerous additions and a few changes. These additions are accessed

via three different ESCAPE sequences followed by another character which specifies exactly what you want.

The first mode to be discussed is the transmission sequences. With these six functions, it is possible to send large sections of data rapidly to an external device. This device could be a printer, modem, or even your 1802.

If data is to be sent to a printer, it leaves via a port on the back of the board. Data is transmited after the printer signals it is ready to receive information. If, on the other hand, data is being sent to a modem or the main computer, it leaves directly through a serial port. While this is happening, the screen is not updated; after all the data has left via the port, any incoming data, old or new, is put onto the screen.

The second mode allows the normally non-printable control characters to be printed. If you type a control character in, the video board interprets it as a function which you told it to carry out. Because of this, it does not store the value of that character in memory.

To get around this, this sequence of characters tells the board that the third character coming to it is to be stored in its RAM, no matter what it is. Besides the importance of getting an additional 32 characters, one also can store numbers below thirty-two in the video board's memory. This is important in making full use of all the graphical characters possible.

The third mode of operation is graphics. In these graphical modes, I believe Netronics made a few errors.

Netronics specifies that there are four different graphical modes:

- standard alphanumeric
- blank (the data is in the memory but it doesn't show up on the CRT)
- wide graphics (11520 or 3840 pixels)
- thin graphics

Actually, there is one additional thin graphics mode:

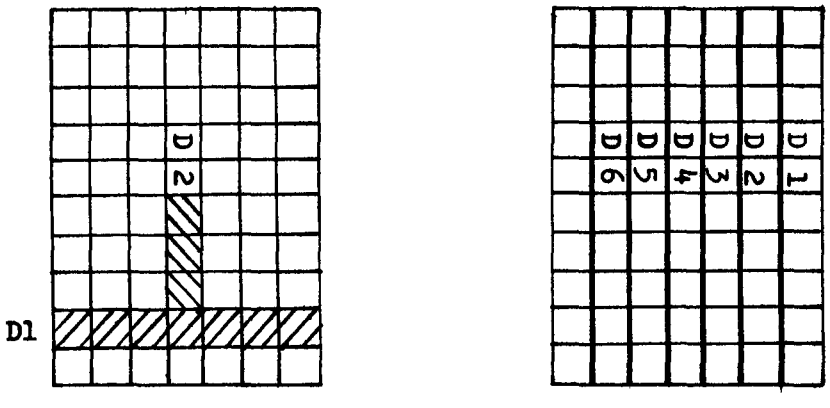


Fig. 1

To enter any of the five printing modes, a sequence of two ASCII characters plus a third one is used. The third character states which of the five modes you want to use.

Eg. Use control @ to enter the vertical line graphics.
Use @ to enter the alphanumeric mode.
Use ` to enter the thin graphics mode.
Etc.

If you look at figures one and two, you will notice that certain regions are labeled with a "D" followed by a number.

This number stands for a binary bit which is allotted to that region for control (The LSB is labeled D0, and the MSB is D6). If a bit in the memory is high, the corresponding region on the graphical character is lit up. If the bit is low, the area, likewise is unlit.

A few problems come up if you wish to store a blank (00 hex) character in memory. A zero in memory will mean that all the regions in the corresponding cell will be unlit. A space cannot be used because the ASCII code for a space is 20 in hex, so some data—lit area—will show up on the CRT.

To store a 00 in the memory, one has to use the correct ESCAPE sequence followed by a control @. Likewise, other graphical characters have to be stored in the same manner.

To add a bit of change to the graphics and printing, Netronics added four attributes—underlining, blinking, reverse video, and half intensity—which are arranged in sixteen different ways. All of these can be used in any of the five different printing modes.

There is nothing wrong about that, but then comes the waste of space. For each of the 16 ways of printing a letter of graphical symbol, there is a corresponding command which will print the attribute but not the actual character or data—ie, blanking.

For example, if you type in the graphics command with an "A" for the third character, it'll print underlined alphanumeric data. If you type in the same command, but instead use a "Q", it'll print the underline but not the data.

If you look at it, and if you realize that this happens for <u>every</u> one of the sixteen attribute combinations and for every one of the printing modes—alphanumeric, vertical, wide, and thin graphics—then you will see that there is a lot of redundancy! Hopefully, Netronics will do something about this sometime in the future.

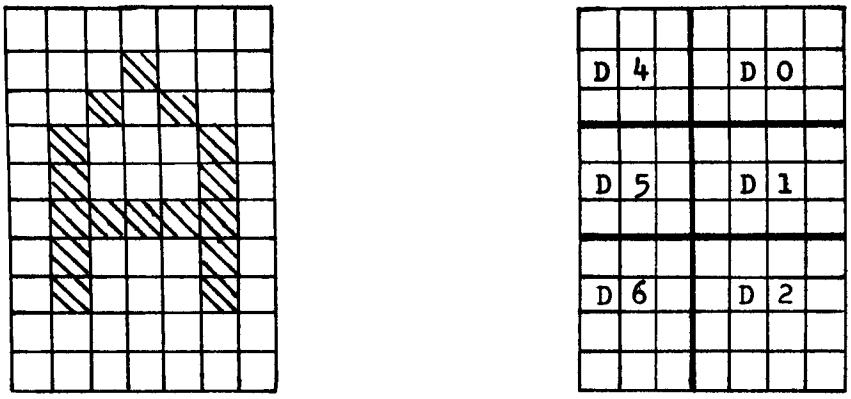


Figure 2

SCREEN CONTROL:

The screen controls in the Smaterm-80 are far superior to the ones on their first video board. Available are the standard move cursor, clear screen, clear-to-end-of-screen, and clear-to-end-of-line. These commands take place immediately after telling the board to carry out the instruction. One does not have to wait—as in the case of their old video board—for the cursor to move to each line or character to erase it.

In addition to these, you also have the ability to insert or delete a single character or more in any given line. If you are inserting a character, everything to the right of the cursor is shifted one space over and a space is generated where the cursor is situated. The opposite of this occurs when you delete a character.

One can also move all the characters below the cursor one line down. In doing so, the line directly below the cursor is all spaces and the bottom line is lost. If, on the other hand, you move all the lines below the cursor up, the present line is replaced by the one directly below it and the bottom line is all spaces.

In addition to that, you have the ability to set or erase a hardware tab at any of the columns on the screen. When the power is started up, these tabs are automatically set at every eigth space.

Because of this automatic setting of tabs, formatted printing is possible without using sophisticated software. Even without the automatic setting, these tabs are a great help in programing.

USER OPTIONS:

As I mentioned earlier, there are numerous switches and other hardware options under the user's control. For example, you can choose the baud rate—110, 150, 300, 1200, 2400, 4800, 9600, or 19200 baud.

Other switching options are: parity, autolinefeed, autoscrolling, screen format (16 by 40, or 24 by 80), and terminal on/off line. Some of these options may be more useful to the user if one installs switches on the back of your terminal for easy changing of baud rate, terminal on/off line, parity, etc.

The terminal off line switch controls data output. If the terminal is on line to your 1802 then all key entries are sent to the CPU. On the other hand, if your computer is off line, then data can be sent to your CPU on only special occasions.

This feature of having the terminal off line is very useful for editing a high level language program. One just has to list the program onto the terminal, and then make any desired changes to it. Then, by following the ESCAPE sequence described under OPERATION, the new edited version of the program can be sent back to the computer's memory. However, there is a problem with this; this will be discussed later in detail.

The difference in screen format primarily depends on what sort of quality moniter you have. Netronic's supplies three models of the Smart Video Board for different resolution CRTs: the High Resolution version (needs a moniter which "has a horizontal sweep circuit that will run at 18.6 Khz"), the Standard version (15.6 Khz scan, 24 by 80 screen format, characters are in a 5 by 7 dot matrix), and the 40 character version (same as the Standard version except can be used with a TV and it has a 16 by 40 screen format).

Obviously, there is a difference in electronics for each version. Be sure to specify which one you wish when ordering or you may have to send back the electronics for the ones that you need.

Parity can be set to the standard odd and even. Also included are marking and spacing parity if the user wishes to use them.

Besides switch selectable options, there are a few hardware options. There is a spot on the board which is reserved for a buzzer or bell which can be turned on via an ASCII control character. Netronics doesn't include this with the kit, but the ROM software is written to accept it if you wish to include this option on your board.

Also are the possibilities of connecting a printer or modem to the board via a RS232-C and serial interface. Netronics' printer—Comet 1—sells for $299.95 (US) and it has the following features: up to 9600 baud rate, bi-directional printing, 80 characters per line or 132 compressed, 9 by 7 dot matrix, and more. Their modem is a standard 300 modem which sells for $189.95.

PROBLEMS:

There are a few technical problems with this board. The major one is not a problem with the actual board, but a problem with the computer.

If you are transmiting data to the 1802 with the terminal off line at 2400 baud, the data isn't read correctly if you are using a software serial reader. The problem is that after your I/O routine gets the first character, it goes off into some other routines for a while to process your input and maybe to store it. After it does this, it jumps back into the I/O loop and looks for another character. While the 1802 was off running some keyboard interpreter, some bits from the next character snuck by so they weren't read. So when your routine reads the next character, it gets maybe 4 bits of correct—but out of phase—data plus some garbage.

This problem doesn't exist in the low baud rate range because the keyboard interpreter has enough time between characters to process the last incoming character. But once you start getting into the thousands of bits per second, synchronization becomes extremely touchy.

The next bug is a problem somewhere on the video board. I have not checked this problem with any othere Smarterm-80s, so I cannot be sure if it just isn't our own one board.

If one moves the cursor up to the top upper right hand corner, and performs a clear-to-end-of-line or clear-to-end-of-screen command, the board seems to shut itself off. The CRT display suddenly goes blank—no data or cursor—and a clean scan shows up.

There is a bit of life left in the board, though. One more character can be entered via the ASCII keyboard and it will be sent to the 1802. I have not successfully found a way of turning it on again via software—the only way seems to be the power switch.

In conclusion, I would like to say that this board is ideal for a person who is interested in a good quality CRT display. It offers a good varity of graphics, fast speeds, and flexibility. Netronics has supplied an area for an additional ROM on this board. This new ROM may make this board faster, it may include more graphics, or, who knows?

- by M. Smith, 1 Cranleigh Ct., Islington, Ont., Canada. M9A 3Y2

How many times have you had to list a whole machine language program just to fine one runaway op-code? While debugging a program, I encountered such a problem lots of times. Because of this, I set out to write an addition to M. Franklin's monitor - SYMON (IPSO FACTO, July, 1982).

Basically, this program looks in a specified memory location and checks if the byte for which you are looking is in that spot. If it finds it there, it prints out the address, and 16 bytes of data and waits for a key press to continue.

The question mark is used to activate the search routine. If the user wishes, the question mark can be changed to something else.

To get the program working, all that is needed is to type in the code for the search command, and then to put the command and address data into the Command Input File at location 005D.

Once the program is running, just type in the addresses and the byte of data, and it will handle the rest.

```
ADDRESS     OPCODES     DISASSEMBLY      COMMENTS
02D0        D40647      SEP #0647        GET PROGRAM SEARCH ADDRESS
02D3        D40704      SEP #0704        R8 = START   RA = LENGTH
02D6        FB0D        XRI #0D
02D8        3AD3        BNZ #D3          GET CHAR. TO SEARCH FOR, STORE
                                         IN RC.1
02DA        8D          GLO RD
02DB        BC          PHI RC
02DC        E8          SEX R8
02DD        9C          GHI RC           COMPARE BYTE IN M(R8) TO RC.1
02DE        F3          XOR
02DF        32EA        BZ  #EA          EXIT TO '02EA' IF SAME
02E1        C4          NOP
02E2        C4          NOP
02E3        18          INC R8           TEST IF DONE (RA=0000)
02E4        E2          SEX R2           LOOP IF NOT
02E5        D4067F      SEP #067F
02E8        30DC        BR  #DC
02EA        F80F        LDI #0F          SET PRINT DUMP LENGTH TO 16
                                         IN RC.0
02EC        AC          PLO RC
02ED        D40663      SEP #0663        OUTPUT ADDRESS
02F0        D40678      SEP #0678        OUTPUT 16 DATA BYTES SEPARATED
                                         BY 1 SPACE
02F3        D40655      SEP #0655
02F6        01
02F7        2C          DEC RC
02F8        8C          GLO RC
02F9        3AF0        BNZ #F0
02FB        3FFB        BN4 #FB          WAIT FOR KEY PRESS TO CONTINUE
                                         SEARCH
02FD        18          INC R8
02FE        30DC        BR #DC           LOOP

COMMAND TABLE
005D        3F02  D0 00                  "?" AT 02D0 HEX
```

Real Time Clock and Program

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

The ACE Front Panel provides a Real Time Clock circuit as part of its function. The National 58167AN is a 24 pin buss oriented memory mapped device providing month, calendar day, week day, hour, minute, second, 1/10, 1/100 and 1/10,000 second in a BCD output. The circuit layout for this device is illustrated on Page 38 of this Newsletter. ACE has memory mapped the clock at address FFC0 to FFD8 and assigned port 2 OUT (62) as the "port assist". ACE chose to implement a port assist on the disk board as a means of preventing the accidental reading or writing of a memory mapped location and of ensuring clean memory map data transfer. The same circuit has been implemented on this circuit. If the user wishes, the port assist may be bypassed for normal memory mapped operation by grounding pin 5 input to the 74LS138 via J6. The clock has a power down, low current mode of operation which is maintained by pulling pin 23 (POWER DOWN) low, which disables all inputs and outputs, except the STAND-BY INTERRUPT. This is the normal state of operation for this chip. When a Clock Read or Write is desired, the chip select on pin 1 is actuated (low pulse) and this same pulse is inverted and used to "power up" the clock on pin 23. There have been no timing problems encountered to date, using this method. To write the clock, i.e., set the calendar, time or alarm registers, the user outputs in a loop a:

```
22 dec stack
62 port 2 out
5n store where "n" is the 1802 register set
   to a memory map address FFC0 to FFD8
```

Reading a register requires an "0n" rather than "5n" instruction in the same loop. The clock offers an alarm feature which is activated by storing the desired alarm period in the appropriate buffer(s) and the Interrupt Output on pin 13 will occur (high output) upon a match. Periods as brief as 1/100th seconds to as long as once a month can be programmed and a specified time and/or date selected.

The software listed below may be employed to set the clock and following that is a sub-routine which can be added to your monitor to read the clock.

SET CLOCK ROUTINE

R9 points to appropriate memory map address in range C0-D8. Each register is set individually.

```
F8 FF B9    set R9 to address
F8 nn A9
F8 xx       set data in "D"
22
62          port assist
59          store
7B          "Q" on
00          quit
```

READ CLOCK ROUTINE

The following subroutine was implemented for SYMON 4, but can be adapted for any monitor. Two entry points are provided: the first at address 0280 displays two lines:

```
CALENDAR     YEAR:  MONTH:  CALENDAR DAY
TIME         HOUR:  MINUTE: SECOND      (24 hour clock)
```

Entry 2 outputs TIME only in above format at address 02D3. The Calendar and TIME values are stored in a buffer addressed by R7 for program use - i.e., compare, disk file dating, etc.

	ADDRESS	CODE	DISASSEMBLY	COMMENTS
>	0280	D40287	SEP #0287	CALL CALENDAR READ
>	0283	D402B6	SEP #02B6	CALL TIME READ
	0286	D5	SEP R5	EXIT
		CALENDAR READ		
	0287	F8FF	LDI #FF	R9.1 TO MEMORY MAP PAGE
	0289	B9	PHI R9	
	028A	F8FA	LDI #FA	SET R7.1 TO BUFFER
	028C	B7	PHI R7	STORAGE OF CLOCK VALUE
	028D	D406A5	SEP #06A5	TEXT OUT PSUEDO "82"
	0290	38	SKP	
	0291	3200	BZ #00	
	0293	F8C8	LDI #C8	SET R9.0 AND R7.0 TO
	0295	A7	PLO R7	ABSOLUTE ADDRESS
	0296	A9	PLO R9	
	0297	F882	LDI #82	STORE PSUEDO YEAR IN
	0299	57	STR R7	BUFFER "82"
	029A	D402A7	SEP #02A7	CALL "MONTH"
	02A0	D406A5	SEP #06A5	CALL "CALENDAR DAY"
	02A3	0D		OUTPUT CR/LF
	02A4	0A		
	02A5	00		
	02A6	D5	SEP R5	EXIT
		READ ROUTINE		
	02A7	D406A5	SEP #06A5	OUTPUT ":"
	02AA	3A00	BNZ #00	
	02AC	29	DEC R9	
	02AD	27	DEC R7	DECREMENT REGISTERS FOR
	02AE	22	DEC R2	NEXT BYTE
	02AF	62	OUT R2	PORT ASSIST
	02B0	09	LDN R9	LOAD VALUE
	02B1	57	STR R7	
	02B2	D40706	SEP #0706	HEX OUTPUT DRIVE
	02B5	D5	SEP R5	EXIT
		TIME READ		
	02B6	F8FF	LDI #FF	SET R9.1 TO MEMORY MAP
	02B8	B9	PHI R9	PAGE
	02B9	F8FA	LDI#FA	SET R7.1 TO BUFFER
	02BB	B7	PHI R7	
	02BC	F8C5	LDI #C5	SET R9.0 AND R7.0 TO
	02BE	A7	PLO R7	ABSOLUTE ADDRESS
	02BF	A9	PLO R9	
	02C0	D402AC	SEP #02AC	GET "HOUR"
	02C3	D402A7	SEP #02A7	GET ":MINUTE"
	02C6	D402A7	SEP #02A7	GET ":SECOND"
	02C9	D406A5	SEP #06A5	
	02CC	0D		OUTPUT CR/LF
	02CD	0A		
	02CE	00		
	02CF	D5	SEP R5	EXIT

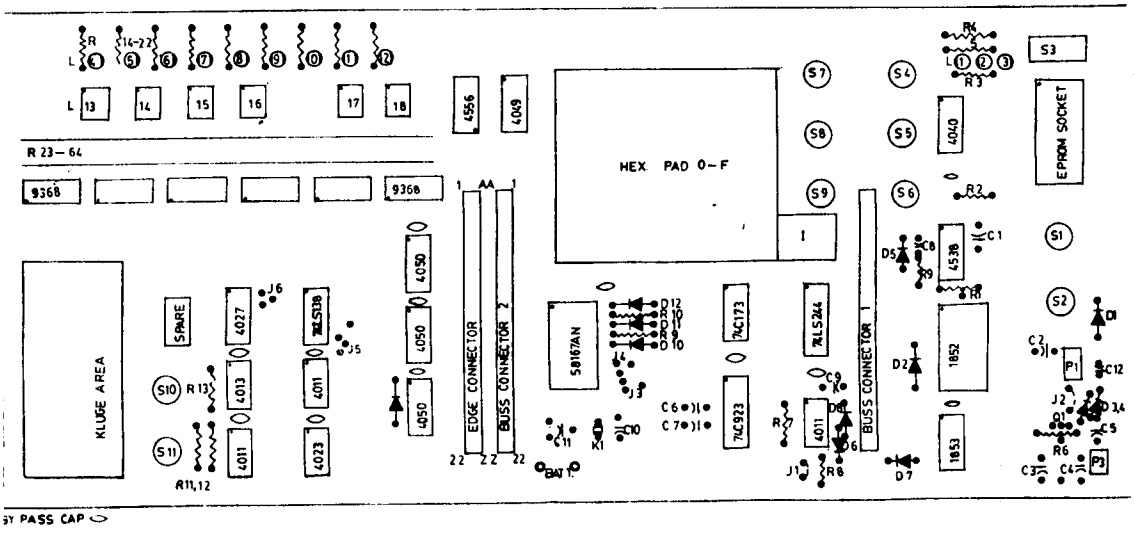
ACE FRONT PANEL

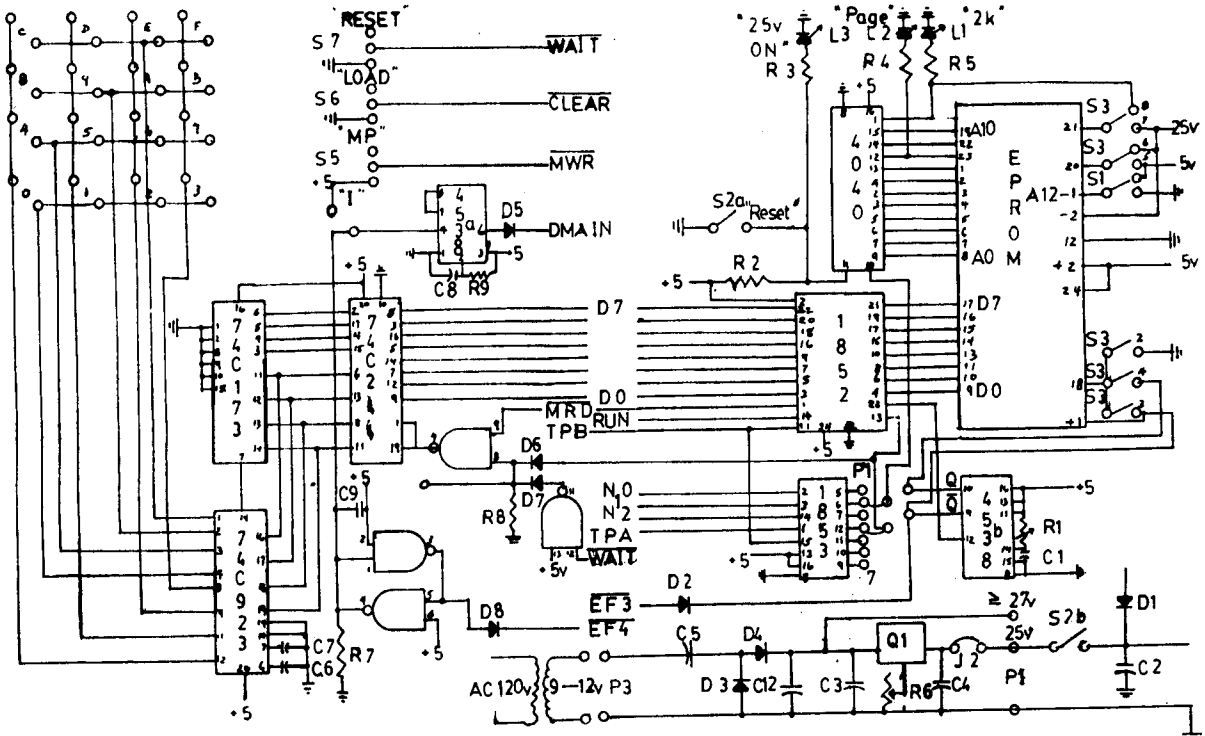
Size: 6" x 13.5"

Function: to provide- a 2716-32-64 Eprom burner (write only)
- micro control switching for RESET/RUN, DMAIN LOAD, MEMORY PROTECT
- Port 4 HEX PAD input
- REAL TIME CLOCK (Nat. 58167AN)
- up front ACE EDGE CONNECTOR
- 4 digit ADDRESS display
- 2 digit DATA display (port 4)
- SINGLE STEP
- PROTOTYPE AREA

Power: +5v, Gnd, +25 to 28v DC for EPROM BURNER

Documentation: Assembly and test instructions, operating guide.
Software for EPROM BURNER and REAL TIME CLOCK.

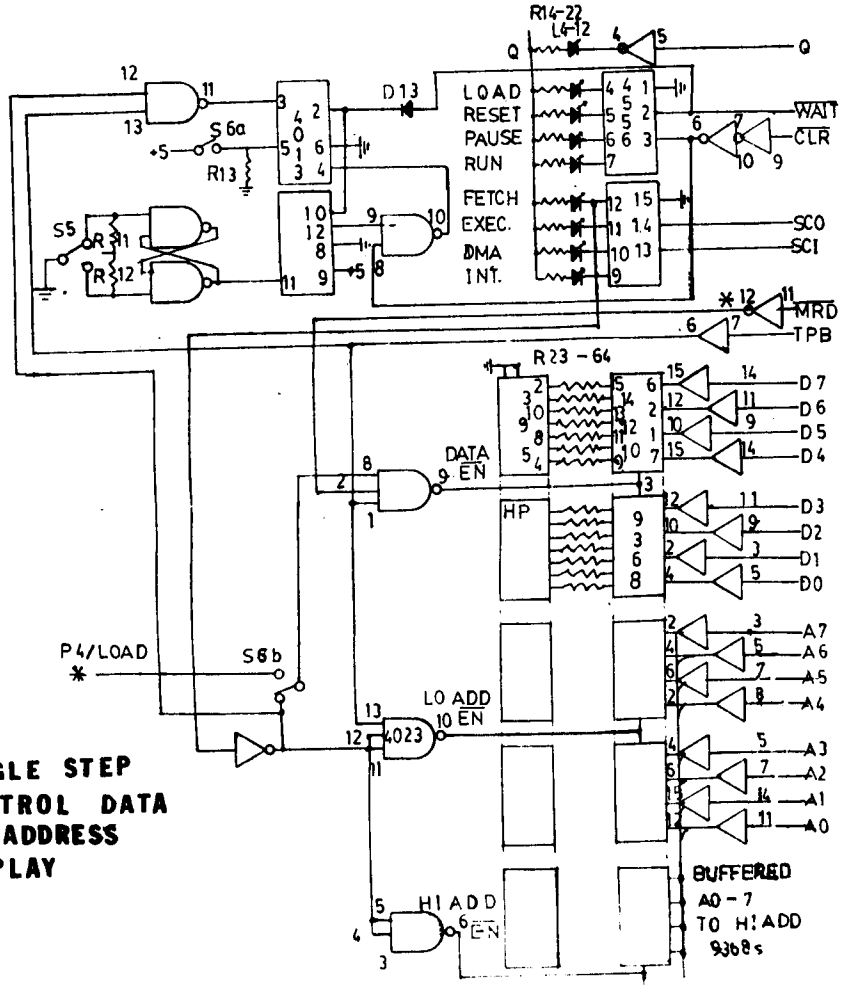




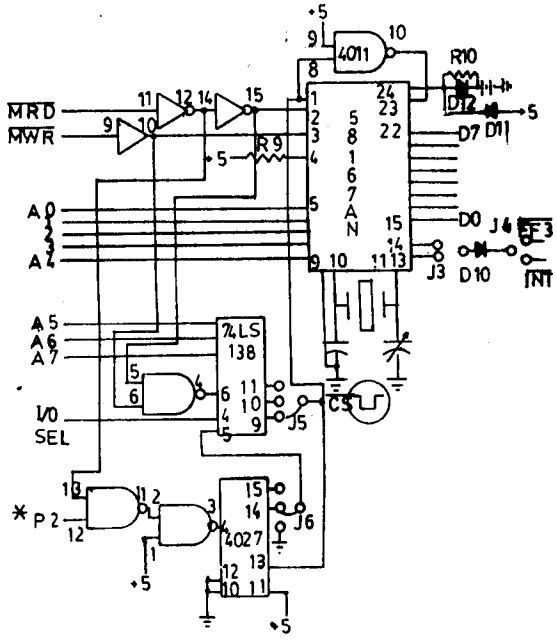
HEX PAD and CONTROL

EPROM PROGRAMMER

ACE FRONT PANEL



SINGLE STEP CONTROL DATA and ADDRESS DISPLAY



REAL TIME CLOCK II

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 format (6K)	_____	$10.00	_____
Tiny Pilot - Netronics Cassette format (2K)	_____	$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.

NAME: _____

MAILING ADDRESS: _____

_____

_____

_____

PHONE NO.: _____

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

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