Ipso Facto

ISSUE 29 MAY 1982

INDEX PAGE

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

1981/82 EXECUTIVE OF THE ASSOCIATION OF COMPUTER-CHIP EXPERIMENTERS

President:	John Norris	416-239-8567	Vice-President:	Ken Bevis	416-277-2495
Treasurer:	Mike Franklin	416-878-0740	Secretary:	Tony Hill	416-523-7368
Directors:	Bernie Murphy Fred Pluthero	416-845-1630 416-389-4070			
Newsletter:			Membership:	Bob Silcox Earle Laycock	416-681-2848
Production Manager: Editors:	Mike Franklin Fred Feaver Steve Carter Bob Siddall Tony Hill	416-878-0740	Program Convener: Tutorial/Seminars:	Bernie Murphy Bert Dekat Ken Bevis Fred Feaver	
Advertizing:	Fred Pluthero	416-389-4070	Draughtsman:	John Myszkowski	
Publication:	Dennis Mildon John Hanson				
Hardware & R. and D.	Ken Bevis Don McKenzie Fred Pluthero Dave Belgrave	416-277-2495	Software: Product Mailing:	Wayne Bowdish Ed Leslie	416-388-7116 416-528-3222

CLUB MAILING ADDRESS:

A.C.E.
c/o Bernie Murphy
102 McCraney Street East
Oakville, Ontario
Canada
L6H 1H6
Phone: 416-845-1630

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 the 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 original, 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 advertisements 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 (ie Netronics Basic only, Quest Monitor required, requires 16K at 0000-3FFF etc.). The newsletter staff will attempt to publish Ipso Facto by the first week of: Issue 25 - Oct 81, 26 - Dec 81, 27 - Feb 82, 28 - Apr 82, 29 - Jun 82, and 30 - Aug 82. 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

In this issue you will find 3 catelogue additions describing new boards now available from ACE. We now have in stock the 64k DYNAMIC RAM BOARD, the 2716/32/64 Eprom-Ram Board and the new 14 slot backplane with on board serial/parallel I/O and Netronics compatible cassette I/O. Also available for purchasors of our Version 1 Backplane, is an I/O adapter offering the same features as the new Backplane as an add-on upgrade of the existing board.

In the works, and scheduled for a July delivery, are a new front panel, with EPROM BURNER, Real Time Clock, hex-led address and date display, HEX PAD and control switches; a new CPU board - not a trainer - configurable for system or dedicated applications; and a new 80-24 video terminal.

To those who had to wait for recent board deliveries, our apoligies. Some boards have great appeal and orders may exceed our limited stock, necessetating a reorder which may take 4 weeks. (We have the lowest delivery priority and hense the lowest price from our board manufacturer). Also, if your wait is unusually long - write - the Post Office doesn't always deliver!!

FORTH

In this issue you will find two articles on FORTH - both good material to get you into it. We also are now offering the 1802 FIG-FORTH on EPROM or Netronics cassette. The program will require patching to your I/O system. Please order documentation and manuals from FIG. ACE does not supply program documentation.

THANKS

A special "thanks" to Richard Cox for his timely and helpful schematics for an EPROM/RAM Board. His work formed the basis for our own board.

Thanks too, to those who wrote about their monitors. Both comments were helpful!

Due to an overwhelming lack of interest; (are you all dead out there?) the 1 vote I received for Best Article for issue 26, 27 and 28 has prompted me to consider cancelling the feature.

Stuck on ideas for projects - how about a multi location temperature sensor, a burglar alarm, a BASIC account program, a daily events calander, a metric conversion program? Happy computing and please write!

1802 MICRO SEMINAR

ASSOCIATION OF COMPUTER-CHIP EXPERIMENTERS
(A.C.E.)

Announces its First Annual Mini-Seminar
to be held on Saturday, August 7, 1982
at Niagara College, Welland, Ontario
(15 miles from Niagara Falls & Buffalo N.Y.)
The keynote speaker of the day will be
Jan King (W3GEY) Vice President Engineering
Amsat Corp. Washington, D.C.

Amsat is the organization responsible for the design, construction and launching of amateur radio satellites. Jan is the chief engineer of this organization and he will be describing the use of the 1802 microprocessor in the satellites and the use of their IPS language in satellite applications. (IPS, is similar to the Forth language.)

RCA, Quest, Netronics and TMSI I have been invited to participate with either demonstration, displays or speakers. Look for complete details in the July issue of IPSO!!.

NOTE: Early registration $10.00. ACE must receive 50 early registrations to indicate sufficient interest, else, unfortunately, the seminar will be cancelled. Over 300 members live within an easy days drive -lets see you there!! Write for details, accommodations guide and a map.

** NEW PRICE !! **
PRE-REGISTER - $15
AT THE DOOR - $20
WE HOPE TO SEE YOU THERE

MEMBER'S CORNER

FOR SALE:

John Beringer, 2729 West Sahara #2, Las Vegas, NV 89102

NETRONICS ELF II - 8K RAM, Giant board (I/O), ASCII keyboard, and modified TV. A lot of software, including games, is included. Asking $450.

BARE-BONES PRINTER - 40 column dot matrix format. Does not have a cabinet. Needs a parallel port to connect, or will connect to above ELF. Software to drive printer already written for above ELF. Asking $200.

VIDEO BOARD/SERIAL KEYBOARD/RF MODULATOR - mostly debugged, with no apparent problems. Asking $150.

A Bosivert, 4830 Des Pervenches, Orsainville, Quebec 7, Quebec, GIG 1R7

Kilobard #27 to #63 @$2.00 each. Some ETI and popular electronics magazines. Send for a list of available magazines.

J. Brainte, 18 Alison Pl, Guelph, Ont., N1H 6X7, (519)-823-5708

Complete ELF II - G.B., 2x4k RAM, Cassette Controller, Documentation Working $200.00, Cdn.

New EDM-926 Electrohome 9" 16MHZ video monitor - $195.00 Cdn.

A. Miller, 2328½ 3rd Ave.,E., Hibbing, Min., USA, 58746 218-263-9131

1-Netronic Fasterm VID Board - New $100.00
1-Netronics Electric Mouth S100 buss, both ROMs $125.00

HELP!

Brent Watkins, P.O. Box 80602, Fairbanks, Alaska, USA, 99708

I would like to have some printed circuit boards made of my 1802 computer. In this part of the country there are no commercial companies to do this. If anyone Knows of a low-priced company that will do small quantities I would greatly appreciate hearing from you. I need boards made that are double-sided, plated-through holes, and with gold edge connector. I supply the artwork.

H.E. Kautz, Jr., 1115 E. Caracas Ave., Hershey, P.A., USA, 170033 (717)534-2642

2-4K Static Ram Boards, Giant Board, color graphics/music board, expansion power supply, RF modulator, fully socketed and professonally assembled. No mods. All Netronics Data with extras all for $425.00 USA funds. Contact the above.

FORTH IMPLEMENTATION NOTES - ACE SYSTEMS

TONY HILL
RR #2
HAMILTON, ONTARIO
L8N 2Z7

As mentioned in the last issue of IPSO FACTO, FORTH for the 1802 is now available and ready to go. In fact, ACE is now distributing copies for essentially the distribution cost. This article is to tell you a very little bit about FORTH, and to explain how to go about bringing it up on your system. Further articles are planned, but for now this should be enough to get you up and running.

FORTH is a programming environment which encorporates both a language, a compiler/interpreter, a "run time" package, an editor/assembler and an operating system. The language is to BASIC as a four function calculator is to a Hewlett-Packard programmable calculator. (This is actually a good analogy as FORTH and the HP calculator both work in reverse polish notation and can seem confusing until you come to appreciate the elegance of the system). FORTH is much faster than interpreted BASIC and offers a much simpler interface with the "real" machine. It would be impossible to describe FORTH in an article twice as long as this, so I think it will suffice to say that it is an interesting language which tends to become addicting. In view of the shortage of other good languages for the 1802, it becomes especially attractive.

Having said that, it's time to explain how to get this "wonderful" language up on your system. First of all, I should point out that there is a well organized group of FORTH users called the Forth Interest Group (or FIG). They have released "standard" FORTH implementations for a large number of systems, including most of the popular micros AND the 1802. I'll talk more about them and the 1802 approved version in future articles. For now it is sufficient to point out that they will sell you a general FORTH implementation manual and a source listing for 1802 systems. You will need both if you intend to get into FORTH seriously, but most especially the implementation manual. It lists the exact definition of all FORTH programming words and is the basic reference for FORTH systems. Note however that it does not teach you to program in FORTH, and at this point I have not read enough of the available books on FORTH to comment on a good basic learning text. The August 1980 issue of BYTE, which was dedicated to FORTH, is not a bad place to start though.

So, to get started, order the implementation manual and the source listing (or get photocopies from a friend, a practice encouraged by FIG) and load the code into your system. You can either type the whole 5-1/4 K in one byte at a time, or buy PROM's or tape from the club and transfer it onto your system that way. (See the end of this article for ordering information for all items mentoned). Then you must customize the I/O to match your system. FIG code includes some sample I/O routines but if you already have a monitor with your own routines callable by SCRT, I would recommend using those. Assuming you are going to do just that, here's how to patch them in. Example code is given for an 8k of RAM system.

FIG-FORTH code occupies memory from 005E to 153C in the basic version supplied by FIG. The user customizes this by adding any initialization code required for his system (usually in the space between 0000 and 005E) and some I/O interface routines. A tested method of doing this is explained below. The other customization required is to allocate RAM space for FORTH stacks and buffers. Usually the top two pages of RAM are used for this. The 16 bit addresses are then stored as follows:

MEMORY LOCATION	ADDRESS OF -	FUNCTION	EXAMPLE
006E/F	Top RAM page	USER variable area	1F00
0070/1	Top page - 1	Computation stack	1E00
0072/3	Top byte of top page - 1	Return stack	1EFF
0074/5	Half way up top page - 1	Terminal input buffer	1E80

INITIALIZATION CODE

Code to initialize SCRT registers, video cursor addresses, baud rates or any system dependant functions can be installed in memory at addresses 0000 to 005D. FORTH actually starts at 005E and so the space below that address is available to you. (Your initialization code should end with a BR 5E). The initialization code should set R3 as the program counter and ACE systems should also load the address of SCRT Call and Return routines into R4 and R5. R2 can be set as the X register if you like, but FORTH will reset it for itself. Note that FORTH itself does not use SCRT, R4 or R5. Sample code is provided at the end of this article.

I/O

There are four I/O routines that the user must supply. They are patched in by storing the start address of each of the routines at the following locations-

MEMORY LOCATION	FUNCTION	8K EXAMPLE
0543/4	character output routine	153D
0573/4	issue a carriage return	1550
055E/F	character input routine	1565
056C/D	test for break condition	157C

The I/O routines are entered with R3 as the PC. All I/O routines end with a SEP RC.

Code to interface to your monitor I/O routines may be added starting at location 153D. Change locations 007A/007B to the address of the next free byte after the end of that code. Also store this value at 007C/007D. In the example listing, this address would be 1585.

Registers 0,1,4,5,6,E,F are not used by FORTH and may be used as required. FORTH uses R7 and R8 as temporary registers only and so they are available for use as well. However on reentry into any I/O routine they may be changed from what they were left as.

R2 is a FORTH stack pointer and that stack may be used if it is cleaned up before exit from I/O routines. The R2 stack is a grow down in memory stack, and is left pointing to the next free byte. This usage is consistant with SCRT techniques. Note that R2 might not be set as the X register on entry to the I/O routines, so do so if you intend to use it as such.

Registers 9,A,B,C,D are reserved for use by FORTH and must be saved and restored if they are used by your monitor's I/O routines. Pushing them on the R2 stack is a good way of doing this.

OUTPUT ROUTINE

The character output routine is know as EMIT in FORTH. Data is passed to it as a byte pointed to by R9. EMIT should increment R9, load the byte then pointed to by R9 and pass it to an output device. It should then decrement R9 three times. See the example listing of EMIT that allows you to patch in the output routine in your monitor.

CARRIAGE RETURN
A routine must be provides to cause your output device to perform a carriage return and a line feed when called. There are no parameters passed to it. Note that if your output device automatically does a line feed when it receives a carriage return, you should modify the patch so that it does not send a line feed as well. An example is provided.

INPUT ROUTINE
The FORTH input routine is called KEY. It has no parameters passed to it. KEY should read a character from the keyboard, increment R9 three times and store the character read in at the memory location then pointed to by R9. It should then decrement R9, and store a 00 there. Again, see the example provided.

BREAK ROUTINE
The FORTH routine that checks for a break condition is called QTERM. It should increment R9 three time, store a 00 at the memory location pointed to by R9, decrement R9 and store a 00 if there is a no break or a 01 if there is a break condition. (If there is to be no break condition, then store a 00 all the time.) The example routine is a dummy break routine that can be used to get your system up initially.

ORDERING INFORMATION
FIG-FORTH information may be ordered from the following address-

FORTH INTEREST GROUP
P.O. BOX 1105
SAN CARLOS, CA.
94070

Prices are $15 in the USA and $18 anywhere else each for the listing or the installation manual. These figures are in U.S. dollars and FIG requires certified checks or money orders drawn on a U.S. bank; or a VISA or MASTERCHARGE number and the expiry date.

ACE is selling fig-FORTH code to its members on three 2716 EPROM's or ELF II format tape at $30 and $10 (Canadian or US) respectively. The intention of the EPROM's is to allow you to read them into your system and use your monitor to move the data into RAM starting at located a 0000. You do not have to have EPROM memory addressed at memory location 0000 and in fact it is not even particulary recommended. See the order page in this issue for our usual ordering information.

WHERE TO GET HELP
Anyone who has problems with getting FORTH up and running, or having any general questions can feel free to write me. I'll answer all letters, and even if I don't know the solutions to your problem I'll try to make suggestions. (I would appreciate a stamped and addressed envelope from any CANADIAN members that write).

I would also be interested in hearing from anyone now running FORTH who has any comments or tips about the 1802 implementation. Future IPSO FACTO articles will include information on how to get the FIG editor and RAM disk simulation running, as well as some neat little tricks and ideas you may find handy. Thanks to Ken Mantei for his FORTH notes, on which this article was based, and without which I would have had a hard time getting FORTH running.

Good Luck ;S (<-- a little FORTH "in joke")

```
;*************************************************************************
;*                                                                       *
;*  FORTH I/O CODE - FOR INTERFACE TO A RESIDENT MONITOR                 *
;*                                                                       *
;*                 THIS CODE IS INTENDED TO PROVIDE AN EXAMPLE OF HOW    *
;*                 TO INTERFACE THE I/O ROUTINES IN YOUR RESIDENT        *
;*                 MONITOR TO FIG-FORTH.  CAREFUL STUDY OF THE REGISTER  *
;*                 USAGE OF YOUR MONITOR IS NECESSARY TO INSURE THAT     *
;*                 ANY OF THE RESERVED FORTH REGISTERS IT USES ARE       *
;*                 SAVED BEFORE THE MONITOR ROUTINES ARE CALLED.         *
;*                 MODIFY THIS CODE ACCORDINGLY.                         *
;*                                                                       *
;*************************************************************************


                            ;************************************************
                            ;*  SAMPLE INITIALIZATION CODE                  *
                            ;************************************************

0000  F8 07    INIT:   LDI     START            ; SET R(3) AS THE PC
0002  A3               PLO     R3               ;
0003  F8 00            LDI     #00              ;
0005  B3               PHI     R3               ;
0006  D3               SEP     R3               ;
0007  F8 XY    START:  LDI     CALL/256         ; SET UP SCRT REGISTERS
0009  B4               PHI     R4               ;      R(4) AND R(5)
000A  F8 XY            LDI     RETURN/256       ;
000C  B5               PHI     R5               ; ( ANY OTHER INITIALIZE
000D  F8 XZ            LDI     CALL             ;   CODE WOULD GO HERE
000F  A4               PLO     R4               ;   TOO )
0010  F8 YZ            LDI     RETURN           ;
0012  A5               PLO     R5               ;
0013  30 5E            BR      #5E              ; JUMP TO START OF FORTH


                            ;*******************************************
                            ;*  EMIT - CHARACTER OUTPUT ROUTINE        *
                            ;*******************************************

                       .ORG    #153D

153D  19       EMIT:   INC     R9       ; SETUP R(9)

153E  E2               SEX     R2       ;
153F  9A               GHI     RA       ; EXAMPLE OF HOW TO SAVE A
1540  73               STXD             ;  RESERVED REGISTER IF USED
1541  8A               GLO     RA       ;  BY THE MONITOR OUTPUT ROUTINE
1542  73               STXD             ;

1543  09               LDN     R9       ; GET OUTPUT BYTE
1544  D4 WX ZY         +CALL   OUTPUT   ; CALL MONITOR OUTPUT ROUTINE

1547  60               IRX              ; EXAMPLE OF HOW TO RESTORE THE
1548  72               LDXA             ;  REGISTER SAVED AT THE START
1549  AA               PLO     RA       ;  OF THIS ROUTINE
154A  F0               LDX              ;
154B  BA               PHI     RA       ;

154C  29               DEC     R9       ;
154D  29               DEC     R9       ; CLEAN UP R(9) FOR FORTH
154E  29               DEC     R9       ;
154F  DC               SEP     RC       ; RETURN TO FORTH INTERPRETER
```

```
                              ;*************************************************
                              ;*  CR - CARRIAGE RETURN OUTPUT ROUTINE           *
                              ;*************************************************

1550  E2            CR:   SEX     R2       ;
1551  9A                  GHI     RA       ; EXAMPLE OF HOW TO SAVE A
1552  73                  STXD             ;  RESERVED REGISTER IF USED
1553  8A                  GLO     RA       ;  BY THE MONITOR OUTPUT ROUTINE
1554  73                  STXD             ;

1555  F8 0D               LDI     #0D      ; LOAD A CARRIAGE RETURN
1557  D4 WX ZY            +CALL   OUTPUT   ; PASS IT TO MONITOR OUTPUT
155A  F8 0A               LDI     #0A      ; LOAD A LINE FEED
155C  D4 WX ZY            +CALL   OUTPUT   ; PASS IT TO MONITOR OUTPUT

155F  60                  IRX              ; EXAMPLE OF HOW TO RESTORE THE
1560  72                  LDXA             ;  REGISTER SAVED AT THE START
1561  AA                  PLO     RA       ;  OF THIS ROUTINE
1562  F0                  LDX              ;
1563  BA                  PHI     RA       ;

1564  DC                  SEP     RC       ; RETURN TO FORTH INTERPRETER


                              ;*************************************************
                              ;* KEY - CHARACTER INPUT ROUTINE                  *
                              ;*************************************************

1565  19           KEY:   INC     R9       ; SET UP STORAGE AREA
1566  19                  INC     R9       ;
1567  19                  INC     R9       ;

1568  E2                  SEX     R2       ;
1569  9A                  GHI     RA       ; EXAMPLE OF HOW TO SAVE A
156A  73                  STXD             ;  RESERVED REGISTER IF USED
156B  8A                  GLO     RA       ;  BY THE MONITOR OUTPUT ROUTINE
156C  73                  STXD             ;

156D  D4 ZX WY            +CALL   INPUT    ; GET INPUT FROM MONITOR ROUTINE

1570  60                  IRX              ; EXAMPLE OF HOW TO RESTORE THE
1571  72                  LDXA             ;  REGISTER SAVED AT THE START
1572  AA                  PLO     RA       ;  OF THIS ROUTINE
1573  F0                  LDX              ;
1574  BA                  PHI     RA       ;

1575  9F                  GHI     RF       ; GET BYTE PASSED BACK FROM INPUT
1576  59                  STR     R9       ; SAVE IT
1577  29                  DEC     R9       ; CLEAN UP STORAGE AREA
1578  F8 00               LDI     #00      ;
157A  59                  STR     R9       ;
157B  DC                  SEP     RC       ;


                              ;*********************************************
                              ;* QTERM - BREAK CONDITION TEST ROUTINE       *
                              ;*********************************************

157C  19         QTERM:   INC     R9       ; SAMPLE DUMMY BREAK ROUTINE
157D  19                  INC     R9       ;
157E  19                  INC     R9       ;
157F  F8 00               LDI     #00      ;
1581  59                  STR     R9       ;
1582  29                  DEC     R9       ;
1583  59                  STR     R9       ;
1584  DC                  SEP     RC       ;
```

Keyboard Bell Circuit

-by T. Crawford, 50 Brentwood Dr., Stoney Creek, Ont., L8E 1Y2

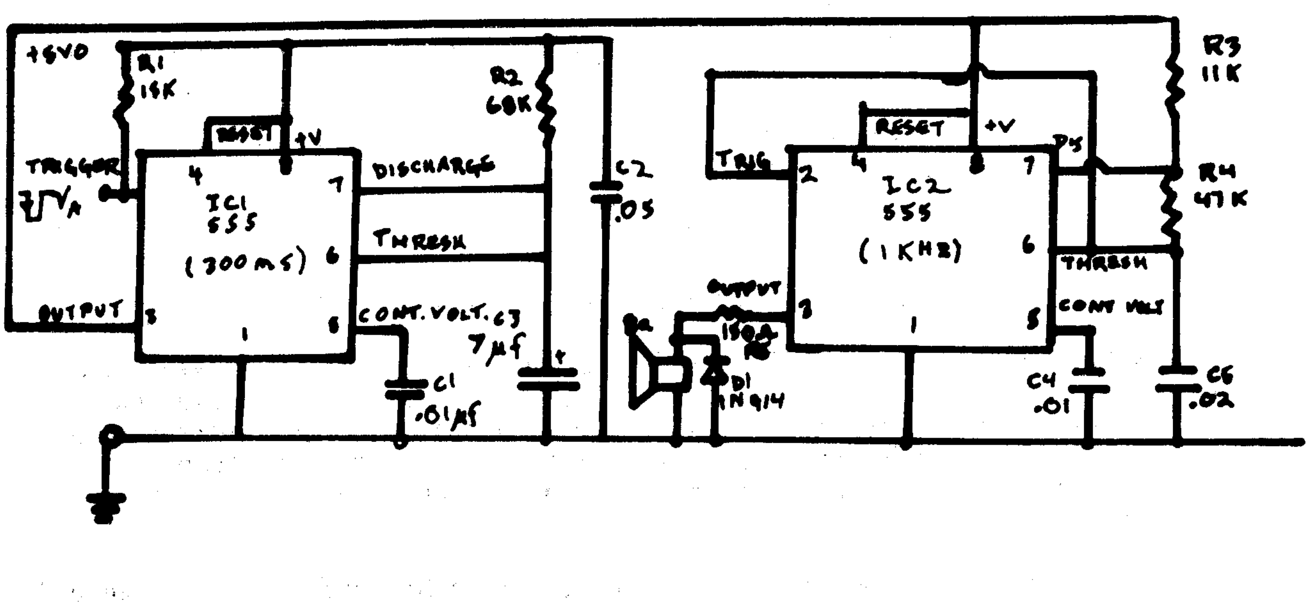
This circuit is designed to provide a short audible tone in response to a momentary negative going pulse (TTL or CMOS) on its input. It is powered from 5V DC, uses inexpensive components, and can be mounted inside a keyboard enclosure.

How It Works

IC1 is connected as a one-shot, with a pulse duration of about 150 milliseconds ($t = 1.1\ R_2C_3$). It is triggered by a negative - going edge at pin 2, the Trigger input of IC1. R_1 provides a pull-up of this input, in case it is left disconnected (as in testing). The Output of IC1, pin 3, is at +5V during the one-shot pulse duration, and is used to supply 5V power to IC2.

IC2 is connected as an astable multivibrator, with a frequeny of about 500 Hz($f = \frac{1.44}{(R_3+2R_4)C_5}$). It oscillates only during the one-shot pulse duraton of IC1, since only then is it supplied with power. The oscillator output is fed through current-limiting (and volume - limiting) resistor R5, to the speaker. This speaker can be any small, 4 to 8 OHM speaker, such as can be salvaged from cheap transistor portable radios. The diode D1 is provided to absorb any reverse voltage spikes generated by the speaker coil. The volume of the resulting tone can be increased by lowering the resistance of R5, and also by connecting to the speaker through an audio output transformer.

Capacitors C1 and C4 are control voltage bypass capacitors, as recommended by the 555 application literature. Capacitor C2 is a power supply bypass capacitor.



MORE NETRONICS, FULL BASIC BUGS

-by B. Erskine, 131 Ave. Adobe, San Clemente, Cal., USA, 92672

I have read several articles commenting on Netronics' Full Basic board with math chip. Several months ago, I purchased a fully wired and tested board with the Basic in EPROM. I found that some of the shortcomings pointed out in the Ipso Facto articles had been corrected but I found one very serious flaw in the interpreter that was not addressed in these articles, to wit, it's inability to pre-assign numeric values to alpha characters entered following an INPUT statement, e.g.:

```
10 LET A=4
20 INPUT X (now type the letter A)
30 PRINT "X=";X
40 END

RUN
X=A (not X=4 as it should be)
```

It required nearly three months and three long-distance calls to Netronics to wring an answer out of them and get the board back. The answer was that the above program would execute properly if revised as follows:

```
10 LET A#=4
20 INPUT
25 A=A#
26 X=A#
30 PRINT "X=";X
40 END
```

It outputted X=4 OK when the letter A was typed but it produced the same result for any other input. As you are probably well aware, TINY BASIC handles this type of program beautifully. This FULL BASIC flaw renders it impossible to write a program to, for example, convert HEX to decimal wherein the decimal values for A,B,C,D,E&F are pre-assigned in the program.

I also discovered a "rounding" error in the scientific notation math mode under certain conditions,e.g.:

```
PRINT TOG 2.2#3* yields 6.E00        (NG)
PRINT TOG 2.22#3* yields 6.666E00    (OK)
PRINT TOG 2.222#3* yields 6.66E00    (NG)
PRINT TOG 2.2222#3* yields 6.6666E00 (OK)
```

Netronics claims to have a Programmer working on this one.

I have been purchasing Netronics Elf II hardware for about four years now and I'll have to say that their user-support is Zilch. Over these years I have never had a single reply to a letter of inquiry (I have written many) with the one exception that I did get a reply to the INPUT statement problem but it was on a piece of scratch paper packed in the shipping container with the board.

Ipso Facto has been a Godsend. Without it I probably would have "given up" in sheer frustration a long time ago.

EDITOR'S NOTE

I have experienced the same problem in lack of communication with Netronics and also with Quest, but to a lesser degree. While both manufacturers sell quite good quality hardware, and at least market some software, there is very little support for the 4000 or so 1802 users.

One item in particular, in my opinion, which is a detriment to advancing the 1802, is the lack of a universally used Monitor or Operating System; such as is available for every other micro system. For this reason, ACE is pursuing development of a Club Standard Monitor.

Winners of the "Name the Program" Contest

-by S. Nies, 134 Four Seasons Dr., Charlottesville, VA., USA, 22901

This article has turned out to be one of the hardest articles I have ever written - to select two names from the excellent names I have received for the Monitor and Text Editor. However, before I announce the names of the winners, I would like to make a few comments (any good emcee has to make a speech, it's tradition!).

By now you may have been wondering if I've forgotten about the contest (I must apologize for missing the March newsletter). No, I haven't forgotten. I was hoping that by extending the length of the contest that more entries would have been submitted. I must admit that at first I was a little disappointed. I figured that even if the software wasn't used, most people would want a free EPROM! I then came to the premature conclusion that perhaps no one liked the 1802 anymore. It seems that quite a few members are leaving the 1802 (and ACE) in favor of larger systems such as Radio Shack's color computer or Apple's apple computer.

However, I feel that most people don't realize that an 1802 based system COULD BE as good as the big, expensive systems. In fact, the two main differences between an 1802 based system and any other system is the amount of software written for the system and the amount of time it takes to execute that software. It is the first difference that determines the usefulness of a computer. A computer is only as good as the quantity (and quality!) of the software available for it.

Perhaps by now it seems that I am in favor of the larger systems. However, the point I am trying to make is that if an 1802 based system had more software written for it, it would be as good (or perhaps better!) as the ones currently on the market.

That's why I have decided to develop software for the 1802 (other than the fact that I hate to throw away a perfectly good system). The only problem so far is that in developing software for a computer, a software foundation must be established. A software foundation consists of a standard software package that can be used to develop more advanced software for a system. This higher level software is then used to develop even more software, and so on.

There are several examples of this process in the marketplace. The most common example is that of the CPM operating system. Way back in the dark ages, before the CPM operating system came along, microcomputer programmers were using their own individual monitors and other software packages to develop software for their systems. This was nice, but not much software was generated at this level. The main reason was that everyone's system was unique. A program written on one system couldn't run on another system, since the hardware configuration was different. Then came CPM. The main advantage of CPM was that it sheltered the user from the hardware specifics. Programs written using CPM as the operating system don't have to know (or even care) about the hardware configuration of a system. Now the marketplace abounds with software that can be run on any system that has CPM.

Unfortunately, the 1802 is still in the dark ages. Several 1802 systems exist, each with different hardware configurations. Software will never be plentiful for the 1802 until a standard operating system is adopted that shields a program from the hardware specifics of a system.

Now for the bright side of the picture. I am in the process of starting a company to develop software (and some hardware products) for the 1802. Since a common medium for software exchange doesn't yet exist, I will be selling the software contained on 2716 (and 2732) EPROMs. All software products will be compatible with ACE, Netronics, Quest, and most homebrew systems. The hardware will be compatible with ACE systems, as well as other systems with possibly a few modifications. The software products planned so far, in more or less sequential order, include the operating system, an enhanced text editor, software to interface disk drives to the operating system, an interactive assembler/disassembler, and a TRS-80 level II interpreter/compiler. All software and hardware products will carry a full money-back quarantee.

So, to sum it all up, I feel that there is no need to abandon 1802 based systems. We simply all need to pull together and start a software explosion for the 1802. Once we start the ball rolling (or flying!!), the 1802 will take its rightful place among the other well known computer systems!

-------------------------- (TAA-DAAA!!) ----------------------------------

And now, the moment we all have been waiting for (for several months!), I would like to announce the names of the winners. As I said before, this really was a difficult task, since all of the names were great!! However, the names of the winners are (drum roll please!):

Wes Steiner for SYSMON (ie. The Monitor)
Bill Swindells for SCRIPTORY (ie. The Text Editor)

Honorable mention goes to:

Robert Decker for EMME (pronounced "emmy"
for Elf Master Monitor Extended)

An EPROM with the software package of your choice will be mailed shortly!

Implementing Quest Tiny Board - enough to make a grown man cry!
-by Wm. Swindells, 1315 Sherwood Court, Burlington, Ont., L7M MK8

There I was, all set to go. I had finally decided on a monitor for my system (Steve Nies Monitor, Version 2). Everything was up and running well, and I was just scooting along in machine code, being able to perform all kinds of neat functions like memory examine, modify, move, save, load, etc; all the neat things the monitor was able to perform. That was all very well, but I was still looking at meaningless hex numbers in long strings on my CRT. What I needed was the ability to put "stuff" on the CRT by means of a "high level language", and allow me a little more expression of thought without having an incorrect hex byte shoot my program off into the unknown, never to be seen again.

Realizing I'm not the smartest individual around, I needed a plan. I would load my 2K of Tiny Basic into memory and execute - that would put a whole new world at my hunt and peck fingertips. **Wrong!!** Tiny went into memory OK, execute went OK, in fact too well - it executed (killed) everything in memory - all gone with a blinding flash from the CRT. Something was wrong for sure!

Upon closer examination of the literature supplied with T.B. (when all else fails - follow the instructions, stupid!), it told me that certain long branch instructions may have to be changed for the I/O routines. What the heck is an I/O routine? How come the darned thing didn't work - it's meant for my 1802, the listing checks out, my monitor works, I guess the only thing left is the "intelligent system" that punches the keyboard - me.

After a few phone calls to Ken Bevis and Bernie Murphy and a couple of hours of sitting with Bernie, looking at the listing and the "I/O" devices of my monitor - I saw a logical pattern developing in the way T.B. was to be interfaced - it sure made sense - Thanks, Bernie.

First, we considered the long branch instructions located near the beginning of Tiny, starting at location 0106 to 010E. The first long branch instruction had to be changed to a routine that would read the keyboard input device within my monitor. The second long branch was changed to the output routine of my monitor and the third long branch was temporarily disabled by using a D5 instruction. I needed some extra space to locate the required routines to move me from T.B. to my monitor and back again.

Since the Quest Tiny Basic did not occupy a full 2K, I had 19 spare bytes left over at the end of the listing. Beautiful - a place where I could locate my input and output call and return routines from T.B. to monitor and back. Tiny occupies 0100 hex to 08E6 hex. The following are changes that were made:

MA M	OP
0106	C008E7
0109	C008F0

Because of the way in which the monitor checks the keyboard for a character input, the following routine was located at location 08E7 hex: D4 C7 5E 3B E7 D4 C7 66 D5. This routine calls the monitor input routine and checks the DF register for a valid character. If the character is not valid, it loops back and looks again. If the character is valid, the character drops through, it is echoed on the CRT and enters Tiny's mumble jumble.

At location 08F0 hex the output routine for Tiny is called by the following: FE F6 D4 C7 66 D5. This simply outputs the characters that Tiny wants to display. The two bytes at the beginning (FE F6) do a shift left, shift right to get rid of an extraneous character generated within Tiny for use with a printer. Obviously, my monitor sits at location C000 to C7FF hex.

My next problem was how to use the break routine. The long branch at location 010C hex was left as is and the bytes from 0766 to 076E changed to the following: D4 C7 5E D5 00 00 00 00 00.

This now allows me to initiate a break in a program once it is running. This also means that any key hit during program execution will generate a break condition. Sometime, I plan to make it selective of one key on my keyboard as I do not have a break key specifically.

Now there were two remaining items to be looked after. Before the colon prompt came up on my CRT, two inverse video ?'s would appear. The byte at location 0111 was changed from 82 to 80, and now I have a clean CRT screen. The last item was located from 011C to 011F hex. At location 011C the bytes 0900 designate the start of user RAM; at location 011E the bytes 00 00 may be changed to designate the end of user RAM, if you have a program located further up in memory that you do not want Tiny to overwrite.

Thanks to the assistance of Ken and Bernie, I have enjoyed operating my system in a "higher language".

Now on to bigger and better things. With the production of the Club Dynamic RAM Board and lots of memory available to us, I would like to investigate a full Basic program for operation within my system.

The moral of this letter is, that after 2 - 3 months of frustration in trying to do things on my own and not getting anywhere, I contacted a member of the Club and resolved my problems in 2 - 3 days. If you have difficulties - the more experienced members in most cases will be able to guide you to a successful completion of your project. That's why the Club exists - to exchange ideas and induce a feeling of comradeship. You are not alone...there's lots of assistance and knowledge available for the asking. I hope this article helps some other "beginner" get started.

New Memory Test Program

-by E. L. Smothers, 5022 Judge Lynn, Memphis, Tenn., USA, 38118

NOTE: the above program is a revision of a program submitted in IF#25. This version runs at any location by imputing appropriate value in hex address 05 and 08.

0000	90	A4	B2	B4	
04	F8	XX	B3		Start address Hi Byte
07	F8	XX	A3		Start address LO Byte
0A	F8	2C	A2		Stack
0D	84	53			
0F	E2	93			
11	52	64	22		
14	E3	84	F3		
17	3A	1F			
19	14	84			
1B	32	29			
1D	30	0E			
1F	7B				
20	83	23	53		
23	3F	23			
25	64				
26	37	26			
28	7A	13			
2A	30	0D			
2C					

-by E. Tyson, 4485 W. Bath Rd., Akron, Ohio, USA, 44313

OPERATION

The MC14411 and the 1.8432 MHz XTAL form a bit rate generator that provide different baud rates for the UART. One baud cycle equals sixteen clock cycles. The UART is selected by a high level signal at either Ni or No such as from a CDP1853 decoder. The MC1488 RS-232C line driver and the MC1489 line receiver provide interface between the RS-232C peripheral and the UART logic. The RTS signal from the peripheral is not connected to the CTS input on the UART. This is because if the peripheral asserted RTS while the UART was transmitting data, the UART would stop transmission in the middle of the character. The user should refer to the RCA CDP1854 data sheet for a more complete description of operation and use.

EXAMPLE PROGRAM

The following program is an example of how to use the UART to echo characters:

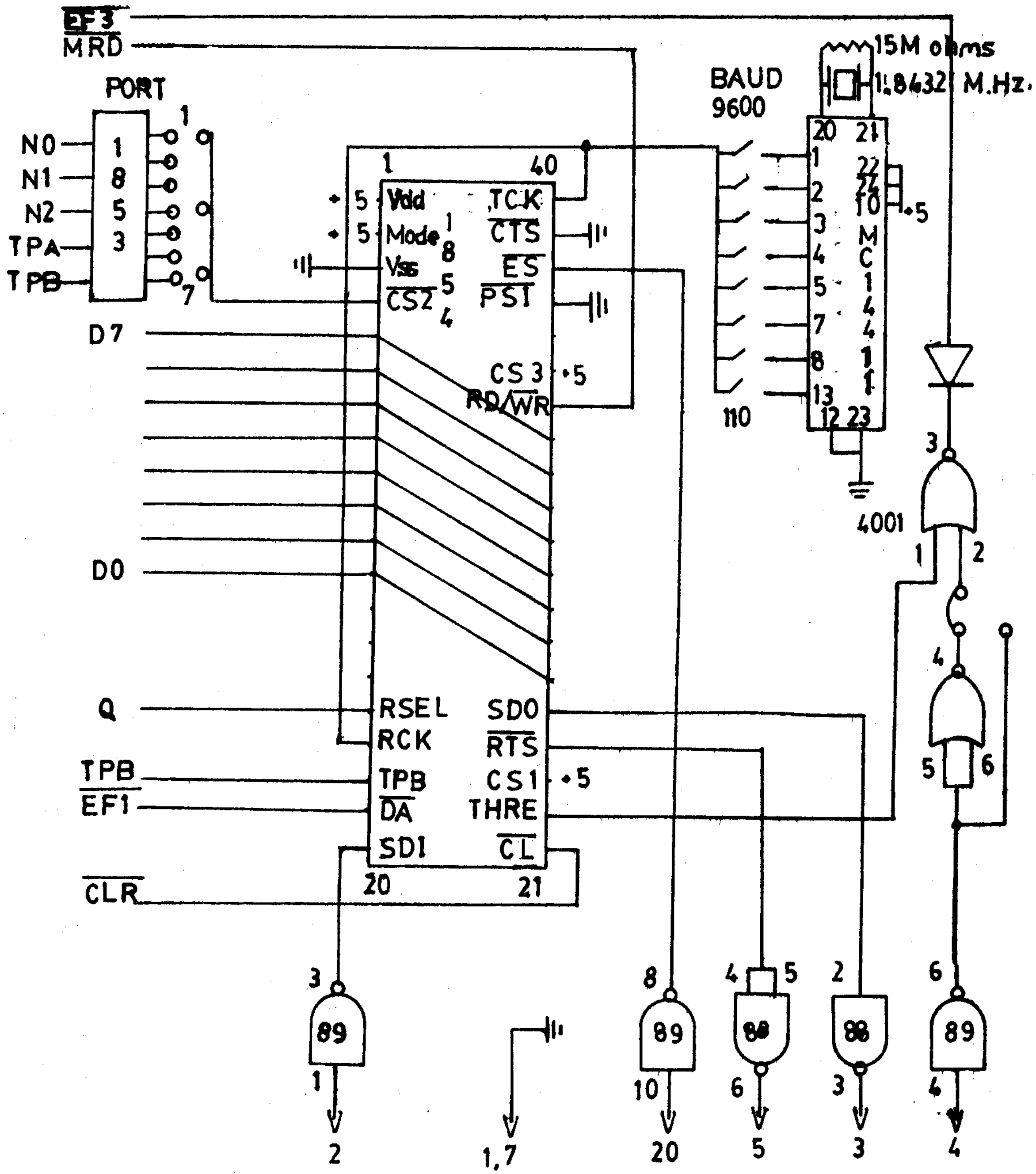
```
0000  7B 66 XX 7A     ..Format UART
0004  F8 01 A2 B2 E2  ..Set up RX as stack
0009  3C 09           ..Wait for data to be available
000B  6E              ..Read byte
000C  36 0C           ..Wait if output device not ready
000E  66 22           ..Output byte
0010  30 09           ..Do it again
```

CONTROL REGISTER BIT ASSIGNMENT TABLE

7	6	5	4	3	2	1	0
TR	BREAK	IE	WLS2	WLS1	SBS	EPE	PI

TR (Transmit Request) normally low
BREAK (Transmit Break) normally low
IE (Interrupt Enable) enables THRE, DA, CTS, and PSI outputs
EPE (Even Parity Enable) high = even parity
PI (Parity Inhibit) high = no parity

WLS2	WLS1	SBS	Function
0	0	0	5 data bits, 1 stop bit
0	0	1	5 " , 1.5 "
0	1	0	6 " , 1 "
0	1	1	6 " , 2 "
1	0	0	7 " , 1 "
1	0	1	7 " , 2 "
1	1	0	8 " , 1 "
1	1	1	8 " , 2 "



EIA RS 232 C DB25 Connector

FORTH- A Bridge Over Troubled Waters

-by L. A. Hart, 366 Cloverdale, Ann Abor, MI., USA, 48105

FORTH(tm.) is impossible. Its proponents claim it is easier to use than BASIC, faster than FORTRAN, uses less memory than assembler, more portable than CP/M, as structured as Pascal, and more fun than Smalltalk. The critics charge that FORTH is primitive, unreadable, hard to learn, and just plain weird. Actually, FORTH can be all of these things and more! In part I of this article, I'll introduce you to this unusual language, and explain what makes it so controversial. Part II will show you an actual implementation of FORTH on the 1802 using 8TH (a version by TMSI) as an example.

But first, just what is FORTH, and why do people say such outrageous things about it? Let's begin with a look at the software gap. This dreaded communications barrier between man and machine has been spanned by countless software bridges, yet we still have trouble getting across. The traditional approach is of course the machine-language path -- all you need is your trusty monitor program, and a keypad and display. This path will eventually get you across, but it's not for the faint-hearted or those in a hurry. Things are a little better if you decide to build your own bridge using an operating system, editor, assembler, and debugger. But it's still a laborious process: Use your operating system to load the editor to create a file for the assembler that you finally test with the debugger. If there are any errors, go back to the editor and start over again. This process is all the worse because each program has its own set of rules; commands, syntax, error messages, etc. keep changing. No wonder the average programmer can only produce about 5~10 lines of good code an hour!

Of course you can use somebody else's bridge if it happens to be going your way. If your computer is large enough and you have enough money, you can do your programming in Pascal, FORTRAN, PL/M or other high-performance language. They substitute a compiler for your assembler, and the rules for programming get simpler and easier to use. But you never get something for nothing - you lose control of things at the machine level (memory usage, I/O, etc.), debugging is much harder, and of course you'll need a disk.

Or you can go to an interactive language, like BASIC. Now things are much easier. You can create, test, edit, and run programs all within the same environment. No more mixed-up rules and constant reloading. However, you now pay a big price in execution speed and memory requirements -- that BASIC interpreter is always sitting there. Besides, there are always the legendary shortcomings of BASIC with regards to standardization and structured design.

"FORTH" is the registered trademark of FORTH, Inc. and anybody caught using it without this kind of footnote or their permission is fair game for lawyers in 3-piece suits.

Enter FORTH: It was created by Charlie Moore in the late 1960's as a better way to program computers (specifically minicomputers). FORTH is a software package that combines the functions of an operating system, monitor, editor, assembler, and high-level language into a single, unified package. It is interactive, like BASIC, and uses one consistent set of rules throughout. It includes a compiler which can produce extremely small, fast programs. Except for a small portion written in machine language, most of FORTH is written in itself. Since FORTH is traditionally supplied with full source code, you can edit and recompile FORTH itself to make any changes you like. The assembler and compiler detect errors incrementally as soon as they are entered, so that you can correct them before proceeding. You can also freely intermix machine language and high level code without difficulty, since the assembler is always available. Perhaps the most unusual feature about FORTH is its extensibility; you can alter or extend FORTH at will to suit your own application or preference. This is the way FORTH is normally used: It tends to grow in the direction the user encourages it to. You generally do not write programs in FORTH; you modify FORTH until it meets the application.

Now if I haven't lost you so far, you can see why it is so difficult to explain what FORTH is: Every version is different! By its very nature, the user modifies FORTH to suit his/her needs. If I'm interested in control applications my FORTH becomes a control systems language. If you are interested in text processing your FORTH becomes a word processor. Rather than fighting change with rigid standardization, FORTH encourages change as the natural order of things. It merely provides rules for change so that any new version will be fully compatible with older versions.

The FORTH philosophy is also a refreshing change from most computer languages. With most languages structure means restriction (you must do this, you can't do that, etc.). BASIC and Pascal in particular were created for beginners and so include many rules, restrictions, error checks, and other techniques to force you to do what their creators thought best. If you know what you are doing, these things just get in your way. It is a little like giving a surgeon a dull knife so he doesn't cut himself. FORTH, on the other hand, emphasizes freedom of choice and creativity, while at the same time meeting all the requirements for modular, structured goto-less programming. You can do the best job only when you are given the sharpest tools and the fewest restrictions. The essential philosophy of FORTH is:

1. The system should have as few rules as possible, and they should be applied without exceptions.

2. The rules should be as simple as possible (but not too simple).

If this sounds like common sense, just try looking at most computer languages! They have so many rules and exceptions that you need college courses to learn how to use them. In contrast, FORTH makes your computer behave more like a pocket calculator; if you know a few rules, you can figure the rest out yourself.

WORDS

So what are the rules of FORTH? First, FORTH is a language of words. Not computer words, measured in bytes or bits, but normal everyday WORDS. In the spirit of minimal restrictions, a word is any string of keys up to 64 characters long with a space on each end. Note that any keys can be used in a word; letters, numbers, punctuation, control-shift-keys, escape sequences -- anything! Therefore all of the following are words:

this THAT 12345 + #!&Z

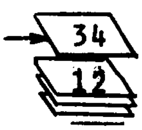
The Dictionary

FORTH keeps a dictionary of all the words it knows along with their definitions. When you type a word on the keyboard, FORTH looks the word up in its dictionary. If it is defined, FORTH does what the definition tells it to do. In essence, the word is the name of a program that gets executed,

The Stack

If you type a word that is not defined in the dictionary, FORTH checks to see if it is a number: If so, it saves it on the Stack. A stack is a convenient place to put things temporarily. (I have several stacks of paper on my desk in fact; the largest being my "IN" basket). The main feature of a stack is that you can only see the item on top, i.e. the last item you put on the stack. Thus if you type two numbers into FORTH, it simply saves them both on the stack, like this:

you type: 12 34 FORTH does this: top of stack →



Since FORTH has its roots in the world of 16-bit minicomputers, each item on the stack is 16 bits (= 2 bytes). Like Tiny BASIC, this makes the largest number +32767 and the smallest -32768. However, almost all FORTH systems support both single-byte and multi-byte precision for great flexibility in handling numbers.

Most of the words in FORTH either leave something on the stack, take something off, or perform some operation on the contents of the stack. For example, "DUP" duplicates the top entry on the stack. "SWAP" interchanges the top two entries. And "+" (remember, anything can be a word) adds the top two entries on the stack, and replaces them with their sum. This system is called Reverse Polish Notation (RPN) and is very much like that used on Hewlett-Packard calculators. It may seem difficult at first, but it actually works out very well. Here are a few examples of math calculations in BASIC and their equivalent in FORTH.

BASIC	FORTH*
PRINT 2+2	2 2 + PRINT
PRINT (13 + 24)/2	13 24 + 2 / PRINT
PRINT (4 + 5)/(6 + 7)	4 5 + 6 7 + / PRINT

The first FORTH example can be read as follows: "Push 2 and 2 onto the stack; add them; and print the result". The second example is "push 13 and 24; add them; divide by 2; and print the result". Notice that parentheses aren't used in RPN -- they are mathmatically unnecessary.

Definitions

Programming in FORTH consists of adding new words to the dictionary. Words can be defined using machine language, the built-in assembler, or in terms of existing words (the high-level language). The system makes no distinction between its own words and yours: They are completely equivalent. In effect, new words extend the language to include new functions. You can also tell FORTH to forget words that are obsolete or unnecessary. It is hard to appreciate the power of this extensibility at first glance. If you define all the key words in BASIC, for example, FORTH _becomes_ BASIC. FORTH dictionaries have been written for BASIC, LISP, and even Pascal!

As an example, suppose you want to define a word "double" that doubles whatever number is on the stack. This could be done by typing:

```
DEFINE DOUBLE               ( define a word named "double")
       2 * ;                ( that multiplies the top of the)
                            ( stack by 2)
```

Comments in FORTH are enclosed in parentheses. Also, FORTH is completely free-format, so extra spaces, line feeds etc. can be used wherever desired for improved readability. The word "define" (usually abbreviated by ":") begins the definition, and the semicolon ";" ends it. To test the word, we can now type:

you type: 3 DOUBLE PRINT

FORTH responds: 6 OK

---

*NOTE: Most FORTH programmers are either hunt-and-peck typists or delight in choosing unreadable names for everything. "PRINT" is thus normally abbreviated "." for example. FORTH listings are hard enough to read as it is, so I've substituted normal words for the cryptic abbreviations in the interests of readability. Part II will then revert to the more common FORTH abbreviations after you've mastered the basic ideas.

Suppose you wanted to write "double" in assembly language so it executes faster. FORTH's built-in assembler is reverse-polish, just like everything else. Keeping this in mind, you would type:

```
CODE DOUBLE          ( define "double" in machine code)
     IRX             ( point the stack pointer to top of stack)
     LDX             ( load top of stack)
     ADD             ( add it to itself)
     STXD            ( push the result back on the stack)
     5 SEP ;         ( return)
```

"Code" begins the definition, and ";" ends it. "Code" has its own vocabulary of words which include all the 1802 assembler mnemonics, as well as the normal FORTH words. Notice the last instruction: You could also have typed "2 3 + SEP" with the same results.

Variables

FORTH has an unlimited number of variable names and types, since you define them as you need them.. The word "variable" creates a new variable as shown below:

```
4 VARIABLE TIME      ( creates a 4-byte variable named "time")
```

The variable is automatically initialized to Ø. In some versions, the number preceeding "variable" is the initial value of the variable and is assumed to need 2 bytes. You can allocate more space if desired like this:

```
Ø VARIABLE TIME      ( create "time",allot 2 bytes,& set to Ø)
2 ALLOT              ( allocate 2 more bytes for a total of 4)
```

From now on when you type the word "time", its <u>address</u> is pushed onto the stack. You can then use "@" (at) to read the value at that address, or "!" (store) to write a new value into it.

```
TIME @ PRINT         ( prints the value of the variable at "time")
100 TIME !           ( sets time equal to 100)
```

The "!" and "@" words have another useful property, too. They can be used to examine and change any memory location, like BASIC's PEEK and POKE (remember, FORTH works with 2 bytes at a time):

```
1234 @ PRINT         ( prints the contents of address 1234 & 1235)
100 1234 !           ( writes 100 into address 1234 & 1235)
```

Control Structures

The "in" thing in computer languages today is "structured programming". I'll not disturb that basket of snakes except to say that FORTH is a fully structured language with no "goto" statement. The most fundamental structure is the "IF" statement. When FORTH sees the word "IF", it pops the top number off the stack and checks for true (≠0) or false (=0). If true, it does every word following until either the word "endif" or "else". The comparison words (=, >, <, etc.) compare two numbers on the stack and return a true-false value for "if" to use. Here are some examples written in both BASIC and FORTH:

```
    BASIC                  FORTH

10 IF 2=3 PRINT 1      2 3 = IF                  ( if 2=3, )
                          1 PRINT ENDIF          ( print 1 )

20 IF A=1 LET B=A      A @ 1 = IF                ( if A=1, )
30 IF A<>1 LET B=0        A @ B !                   ( then B=A )
                          ELSE 0 B ! ENDIF          ( else B=0 )
```

The BEGIN - UNTIL words keep repeating everything between them until the top of stack is True at "until". This is how you build loops that repeat an indefinite number of times:

```
    BASIC                      FORTH

10 PRINT A                 A @ BEGIN         ( begin with A on )
                                             ( the stack)
20 LET A=A-1                 DUP PRINT       ( print A)
30 IF A>0 GOTO 10            1 -             ( decrement A)
                             DUP 0 = UNTIL   ( repeat until A )
                                             ( not true)
```

The DO - LOOP words repeat everything between them a specific number of times. The word "DO" expects the "from" and "to" values to be on the stack. It removes these values and saves them on a second stack called the "Return" stack. The "to" value from the top of the stack becomes the "limit", and the "from" value from the next entry on the stack becomes the "index". After all the words between "do" and "loop" have been executed, "loop" adds 1 to the index and compares it to the limit. If the index is less than the limit, the loop is repeated; if the limit has been reached or exceeded, the two values are removed from the return stack and the next word (following "loop") is executed. Here is the same example program above but using BASIC's FOR - NEXT statements and FORTH's DO -LOOP words:

```
    BASIC                  FORTH

10 FOR I=0 to A        A @ 0 DO            ( DO for I=0 to A:)
20 PRINT I               I PRINT           ( print I)
30 NEXT I              LOOP                ( loop until done)
```

Note that both programs are easier to read. FORTH's DO - LOOP has its own index variable, I, which is completely separate from any of your variables. This means there are no problems with nested loops. Variable step sizes can be used by replacing "loop" with "+loop": instead of always adding 1, it adds whatever is on top of the stack before deciding to loop or continue.

The Editor

FORTH systems usually have what is referred to as a "screen-oriented editor". A "screen" is a block of 1024 bytes or characters, arranged as 16 lines of 64 characters each. This is a bit much for 1802 systems using the 1861 or 1869/70 chip sets, but it works well with simple terminals like the Netronics or Xitex boards. In essence, you load a "screen" into memory and also display it on your CRT screen. The editor lets you edit the screen as desired, and then either save it or compile it for execution. You can have more than one screen in memory at a time and quickly switch back and forth between them. All the rest of the screens not currently in memory are on your mass storage device (disk, cassette, or whatever).

The Operating System

A well-designed FORTH package includes its own operating system. This includes full control over the keyboard (what control-key does what, etc.), the video display (cursor control, etc.) and the mass storage device (disk, tape, etc.). There are also a large number of useful features that give FORTH great versatility.

For instance, FORTH can handle numbers in any base. There is a variable (called "base", logically enough) that sets the number base for all number inputs and outputs. Certain commonly-used bases have their own word so it is easy to change between them, like hex and decimal. Thus you can use FORTH as a number converter like this:

<u>you type</u>:	DECIMAL 15 HEX PRINT
<u>FORTH responds</u>:	F OK
<u>you type</u>:	HEX ØFF8 DECIMAL PRINT
<u>FORTH responds</u>:	2552 OK

You can add new bases either by directly manipulating "base" or by defining a new word to set "base" whenever desired. Let's do the latter:

<u>you type</u>:	DEFINE BINARY 2 BASE ! ;	(define "binary") (to set "base"=2)
<u>FORTH responds</u>:	OK	
<u>you type</u>:	DECIMAL 10 BINARY PRINT	
<u>FORTH responds</u>:	1Ø1Ø OK	

Internal Operation

FORTH has a few surprises in its internal operation as well. The machine-language nucleus creates an artificial CPU (called a pseudo-machine); then all high-level programming is done with the pseudo-machine's more powerful instruction set. This is the same approach taken by the tremendously-successful UCSD Pascal. When you use the assembler you are actually changing the instruction set of this pseudo-CPU.

All this is tied together by a technique called indirect threaded code to produce incredibly small programs. The entire FORTH system usually resides entirely in under 8K bytes of memory - editor, assembler, compiler, and all! FORTH can actually produce programs that are smaller than well written machine language (ah-but once you've seen how FORTH does it you can shrink your machine language programs by the same technique). Execution speed is also very good - about 1/2 that of machine code.

This all combines to make FORTH highly transportable. FORTH is up and running on virtually every 8-bit and larger CPU on the market, and they can all trade high-level definitions freely. Definitions written in assembler obviously run only on machines with the same CPU, but FORTH helps even there. Register usege, memory mapping, I/O, and parameter passing are so well-defined that it is easy to write transportable machine-language definitio tions. And since FORTH gives you full access to source code, you can always modify your version as necessary. In essence, FORTH creates a known environment where everybody knows the rules - meet the rules, and your program runs in any system!

Availability

Now for the bad news. Real FORTH is sold only by FORTH, Inc. for over $5,000. They have a toy version called picoFORTH to demonstrate it, but even it sells for $195. So the guys in the FORTH Interest Group (fig) decided to write their own version for all the popular CPUs. They call it figFORTH and while its much bigger and slower and missing many of the features, it is still a good deal at $15 for the listing. Fig doesn't sell machine-readable code, but some of their members do. I've seen versions by Gordon Fleming, Gary Bradshaw, Richard Cox, and Peter Van Roy.

Technical Micro Systems, Inc. has a version called 8TH that's closer to the original FORTH. It's sold in ROM for $100 and runs in as little as 256 bytes of RAM. This is the version I'll be describing in the next part. Till then, you might look up some of the following references for further reading.

FORTH references:

1. Dr. Dobbs Journal, number 59, September 1981, special FORTH issue

2. BYTE, August 1980, special FORTH issue

3. FORTH Interest Group, P.O.Box 1105, San Carlos, CA 94070

4. Using FORTH, book by FORTH, Inc., 2309 Pacific Coast Highway, Hermosa Beach, CA 90254: Best reference book ($25)

An Inexpensive EPROM Eraser

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

In the last issue of Ipso Facto, I presented a circuit and program to "burn" 2716 EPROMs (+5V versions).

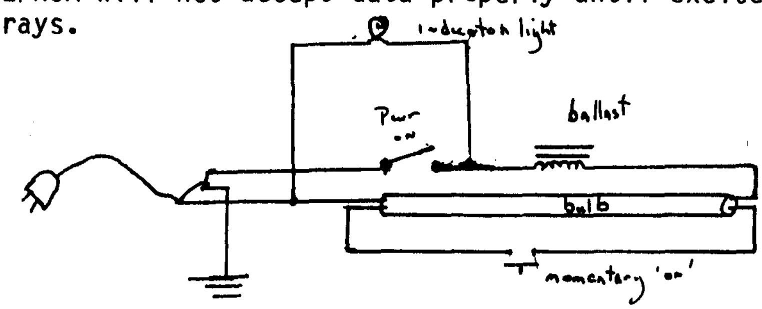
In order to get the most out of your "burner", you will need an eraser. The following circuit is a modification of the circuit presented by Simon in March, 1978 kiloband (p.90).

Parts are relatively inexpensive - you will need an ultra violet bulb such as GE G8T5 which costs about $12.00 from electrical suppliers, and a ballast such as the GE 89G489, costing about $5.00, and 1-momentory and 1-SP5T 120 volt switches.

The GE G8T5 bulb is 12 inches long. I mounted the bulbs on wooden supports for isolation in a 17"X4"X4" Hammmond box and hinged the lid with a piano hinge. The lid becomes the base with the box continaing the bulb, opening upward to expose the EPROM cushion. I glued black felt over the base of the lid and covered the box corner holes to prevent light leaks (the ultra violet rays are dangerous to your eye sight) and glued 8 inches x 2 inches of anti-static foam onto the felt is a cushion for the EPROMS. The attached circuit has been in use for about 2 years without malfunctioning.

I have found that 35 minutes will erase all 2708 and 2716 EPROM brands that I have used.

As a final comment, I have found it advisable to always erase a new EPROM prior to its first use to insure proper "burning". Occasionally, a new EPROM will not accept data properly until excited by the ultra violet rays.



NOW AVAILABLE - ELFISH - AN INTERPRETIVE DEVELOPMENT SYSTEM

```
 0000 0000
>0002 0000<A
 0004 0000
 0006 0000

 REPLA F0F8
```

This 2K package was written for 4K and larger Super Elf's and Elf II's. The package is page relocatable and can reside in ROM. It contains a 1K editor designed to make program entry and modification easy, as well as a 1K interpreter. The editor displays 4 lines, each consisting of a two byte address followed by two bytes of memory content as is shown on the right. REPLA in the diagram indicates that the editor is in the replace mode; successive two byte instructions are entered and replace the memory contents at the indicated line. Other commands are insert, delete, go to, scan up, scan down, execute, assemble, and change address mode (absolute or relative). The interpreter features 16 bit variables and 64 ASCII display symbols; interpretive code is fully relocatable.

A 32 page booklet which includes annotated hex dumps of both the interpreter and the editor, together with a sample program, and instructions for use is available for $6.00 from the address below. Also available are casette tape versions in either Super Elf or Elf II format (please specify) for $6.00 postpaid. A special price is offered when both the booklet and casette tape are ordered at the same time, $10.00 postpaid. Further information can be obtained by writing to:

Paul Moews
34 Circle Drive, RFD 3
Willimantic, CT 06226 USA

Netronics Circuits Mods

-by J. Swofford, 2302 N. Fairview Ave., Decater, Illinois, U.S.A. 62526

Here are two circuits which might be of interest. Figure 1 is a Quest/Netronics cassette control converter. Since Super Basic and the Netronics Text Editor have inverted cassette control bits, this circuit can switch control from one to the other.

Figure 2 is a means of decoding the Netronics monitor at F000 - F0FF only so that F100 FFFF can be used (normally, the monitor is echoed every ¼ K all the way from F000 to FFFF).

This circuit adds a 74C02 and a 74 C30. The previously unused portion of the giant board's A8 (74C174) is utilized.

One note about the Netronics electronic mouth. Before its initial use, check to see if Out 3 and Out 7 (Pins 57 and 53, respectively) is wired to the slot for the mouth on the Elf II buss. Otherwise, everytime the data buss is used, strange noises will be heard.

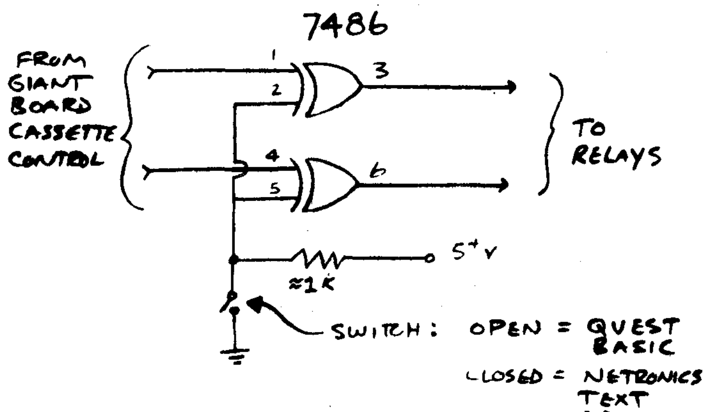


FIG. 1

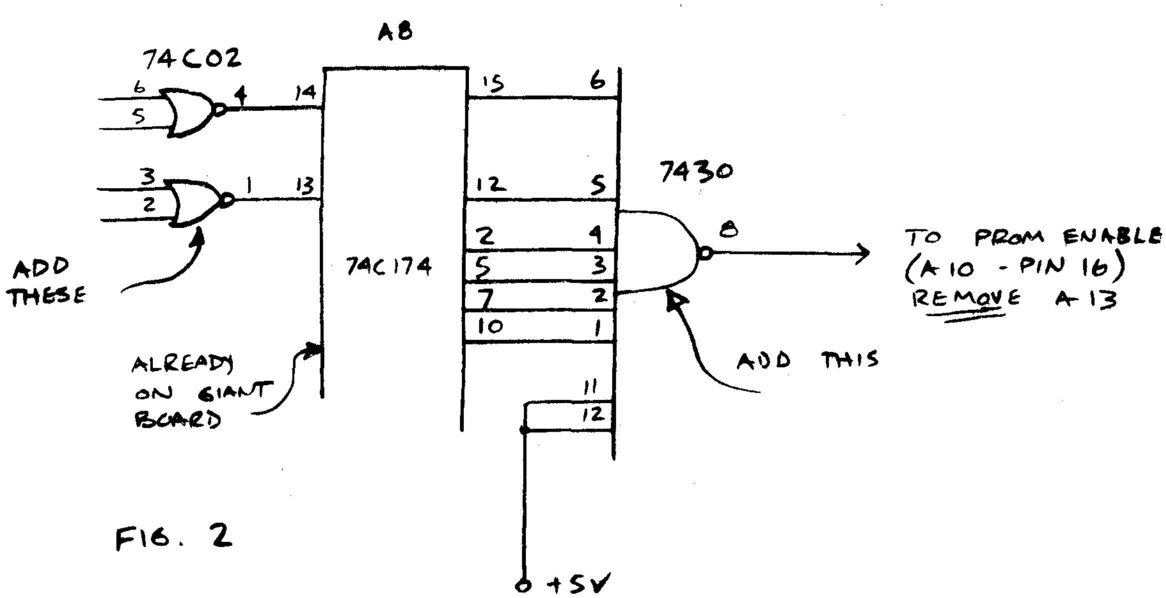


FIG. 2

ACE BACKPLANE AND I/O BOARD.

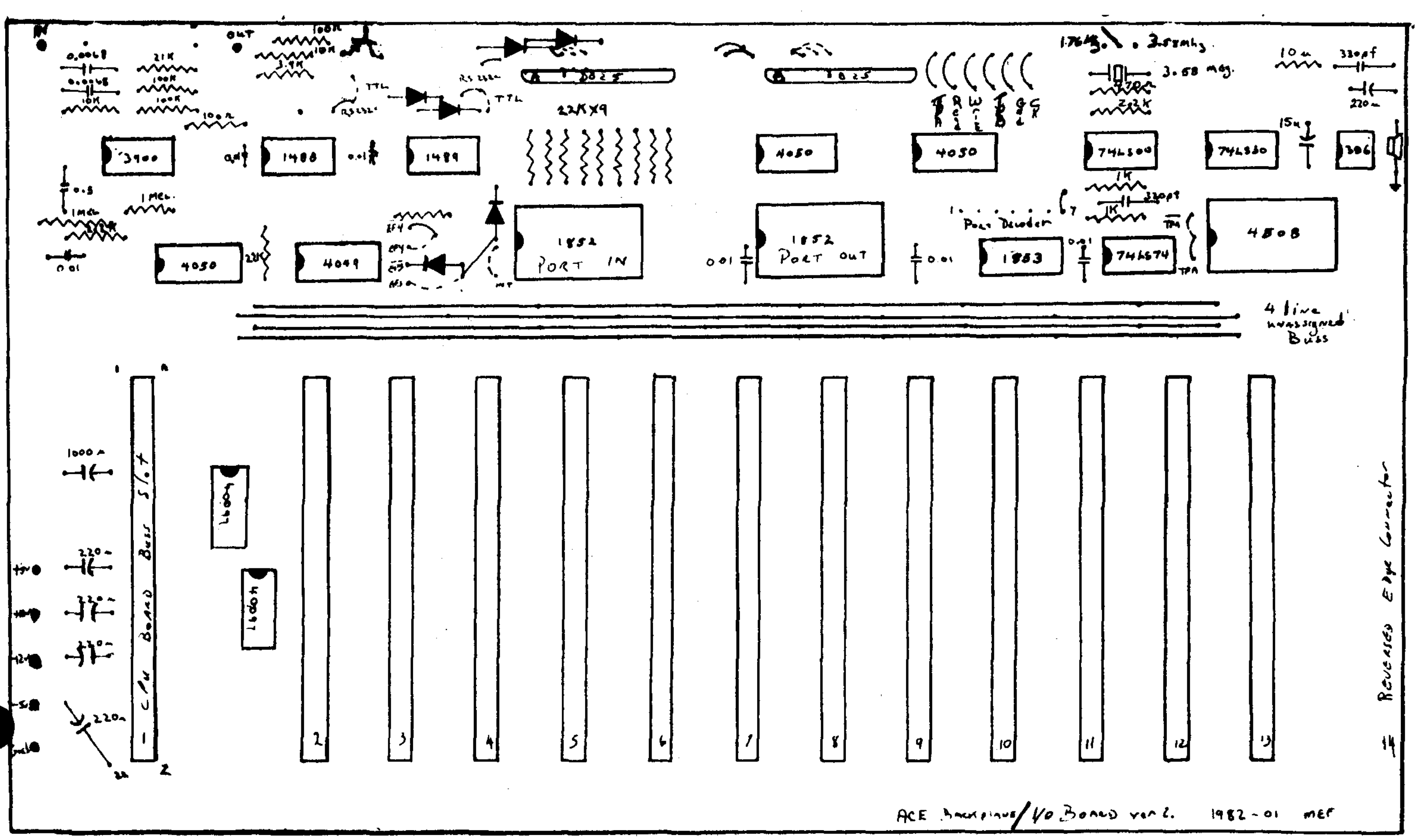
Sixe: 7.0" x 13.5"

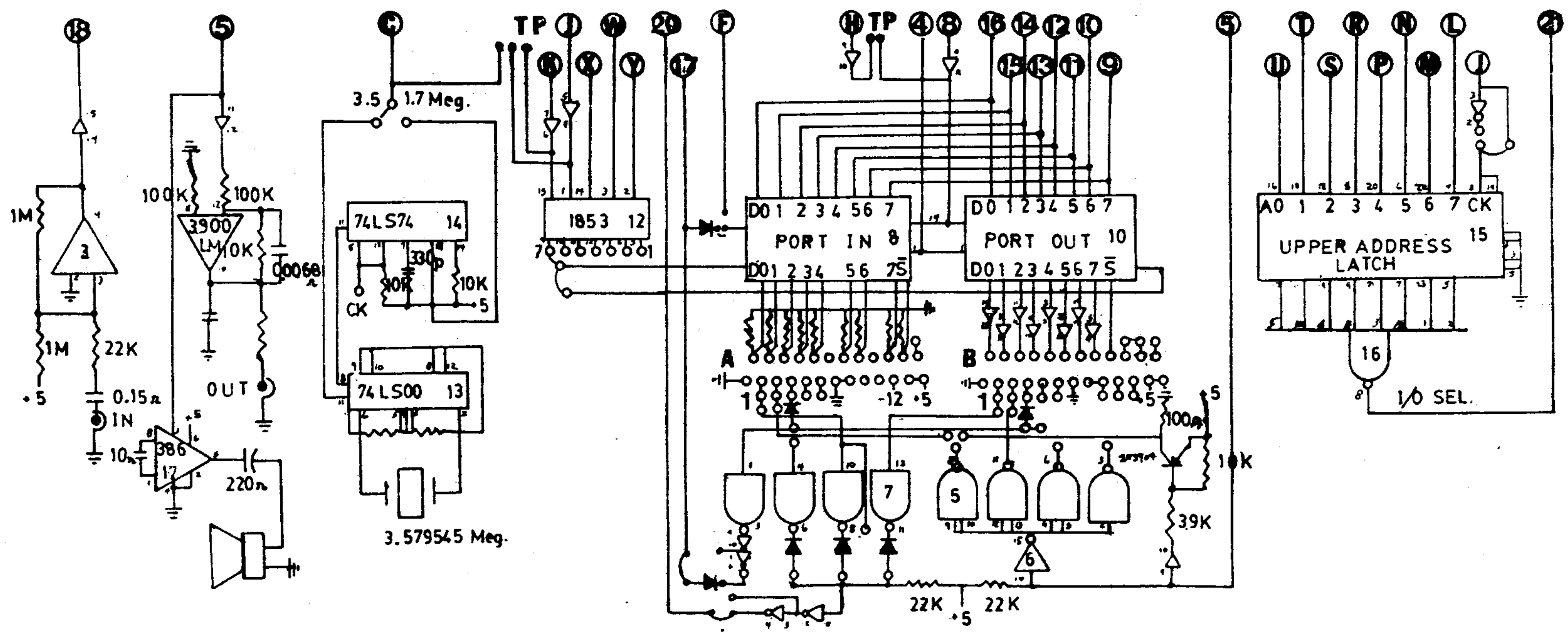
Function: to provide a 14 slot 44 pin motherboard, configured in the ACE standard, with address, MRD mad MWR, TPA and TPB buffered.
: to provide Netronics compatible CASSETTE I/O.
: to provide TTL and/or RS 232C SERIAL I/O.
: to provide PARALLEL I/O.
: to provide a CPU CLOCK
: to provide a MEMORY MAP (I/O SEL)
: to provide a buss power filter and distribution point.

Power: -5v, -12v., Gnd.

Documentation: assembly and option guide.

NOTE: ACE I/O Adapter Adapter Board is available for owners of previous Backplane (with cassette relay controller) which provides the above I/O features as an add-on upgrade to the board. The Adapter is identical to the above board I/O section, and connects to the buss by wire jumpers. The board mounts on the top of the origional backplane by stand offs and bolts. Size: 3.0" x 13.5".





Q-AMP and CASSETTE CLOCK PARALLEL and SERIAL CIRCUIT 'FF' MEMORY MAP

ACE BACKPLANE and I/O BOARD ver 2

ACE 64k DYNAMIC RAM MEMORY BOARD.

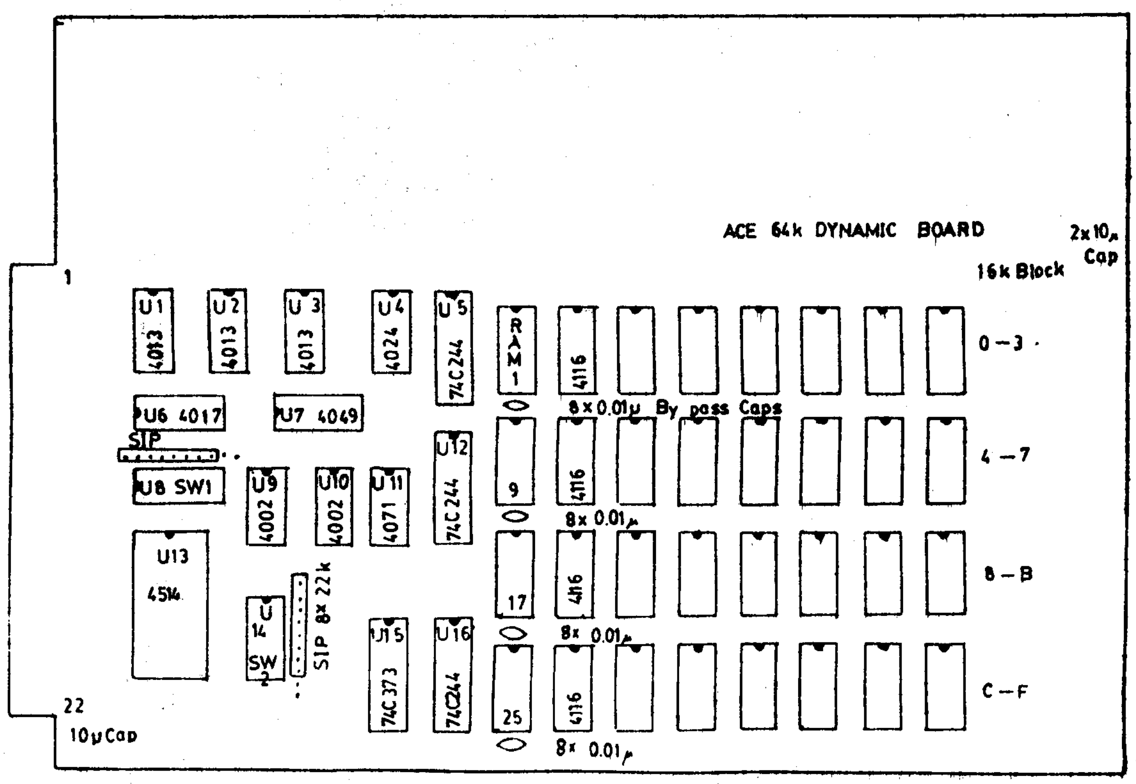
Size: 6.0" x 9.5"

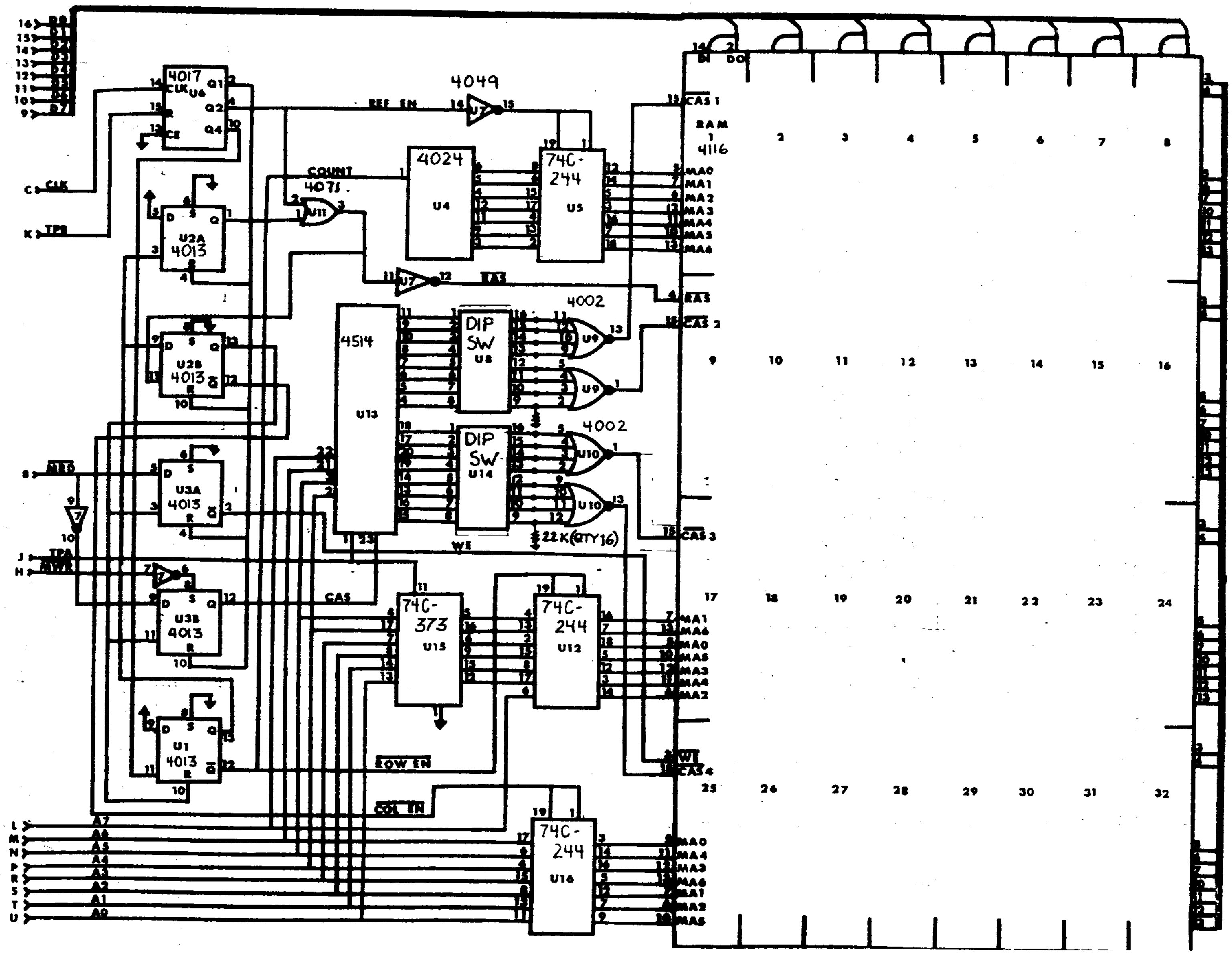
Function: to provide up to 64K of user RAM on the ACE configured buss. On board refresh independant of micro clock. RAM may be disabled in 4k blocks by sue of switches (S 1 and 2). May be populated in units of 16k. Flexible jumper provision at edge connector allows reconfiguration to other 44 pin configurations, ie VIP' RCA Micro board.

Power: -5v, -12v, Gnd.

Documentation: assembly instructions, trouble shooting guide, memory test program, operation instructions.

Cost of complete board (64k) - approximately $125.00.





D0
D1
D2
D3
D4
D5
D6
D7
4017
U6
CLK
CE
Q1
Q2
Q4
CLK
TPB
REF EN
4049
U7
4024
U4
74C-244
U5
COUNT
4071
U11
U2A 4013
U2B 4013
U3A 4013
U3B 4013
U1 4013
RAS
4514
U13
DIP SW
U8
DIP SW
U14
4002
U9
4002
U10
22K (QTY 16)
WE
MRD
TPA
MWR
CAS
74C-373
U15
74C-244
U12
ROW EN
COL EN
74C-244
U16
A7
A6
A5
A4
A3
A2
A1
A0
RAM
4116
DI
DO
CAS 1
CAS 2
CAS 3
CAS 4
MA0
MA1
MA2
MA3
MA4
MA5
MA6

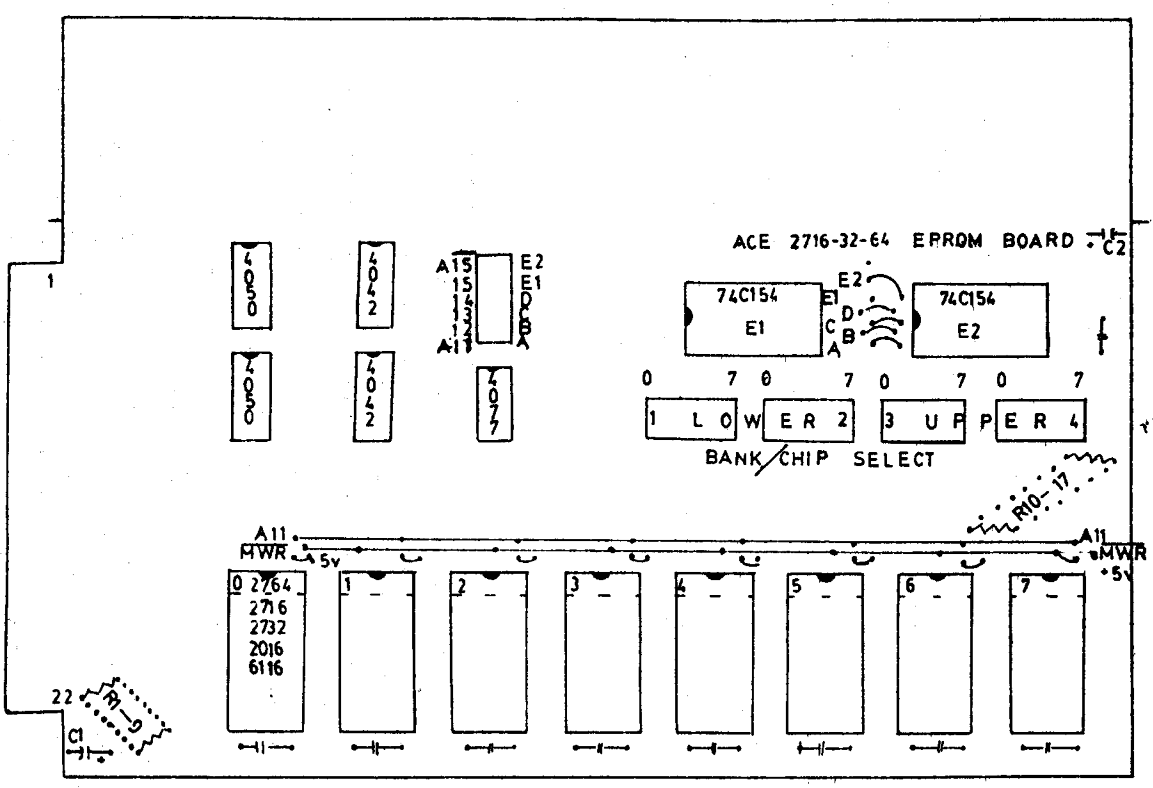
ACE 2716/32/64k EPROM BOARD.

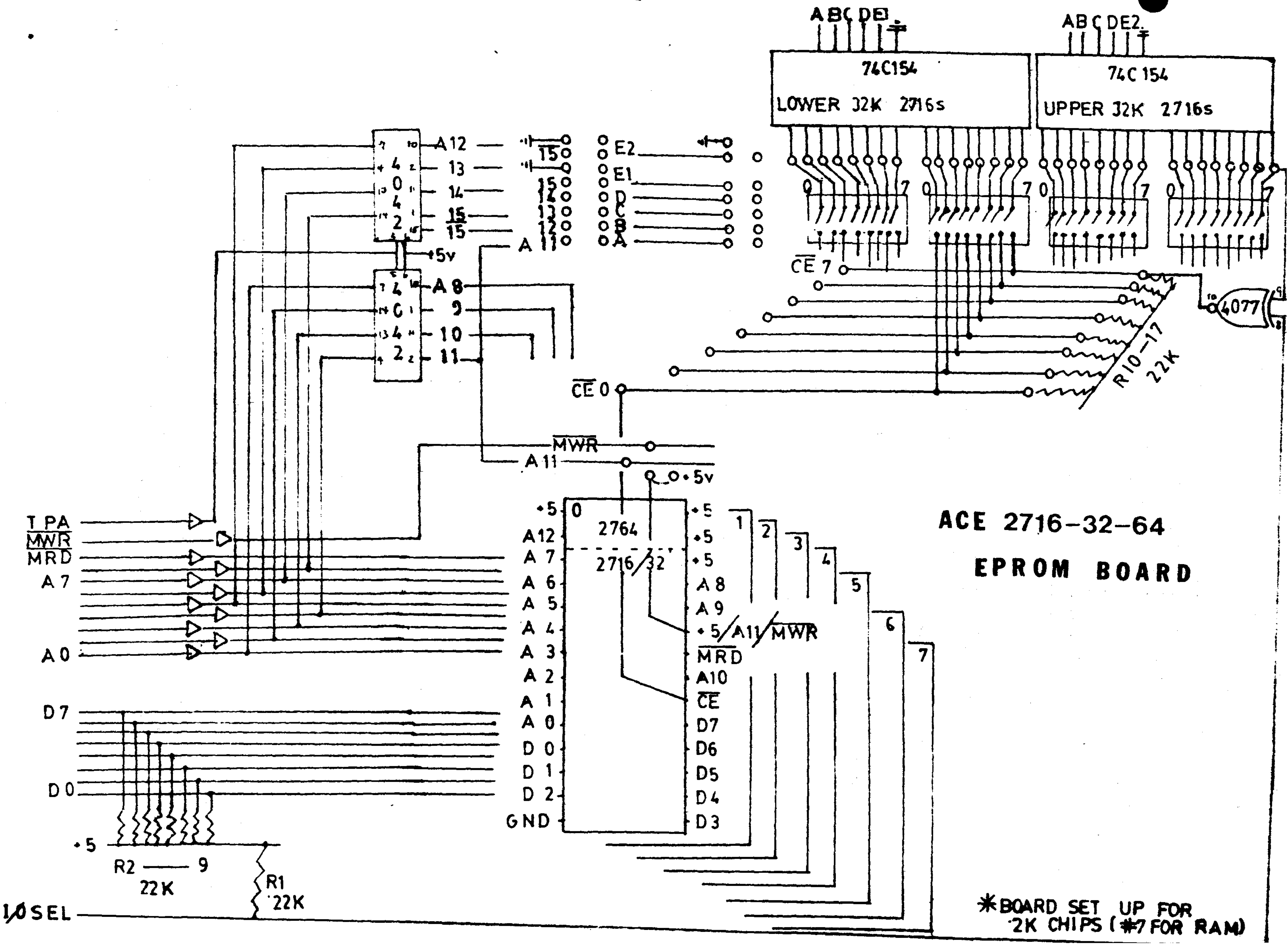
Size: 6.0" x 9.5"

Function: to provide 8 - 28 pin sockets optionally configurable to accommodate 2 - 4 - 8 k EPROM or RAM chips. Decoding allows for location of memory at any location in memory. Two decoders allow mixing of any 2 sizes of memory. On board MEMORY MAP shadow .

Power: -5v, Gnd.

Documentation: assembly and operation instructions.





ACE 2716-32-64
EPROM BOARD
74C154
LOWER 32K 2716s
74C154
UPPER 32K 2716s
A B C D E1
A B C D E2
4042
A12
13
14
15
A8
9
10
11
+5v
E2
E1
D
C
B
A
A 11
CE 7
CE 0
4077
R10—17
22K
MWR
A 11
+5v
TPA
MWR
MRD
A 7
A 0
D 7
D 0
+5
R2 — 9
22K
R1
22K
I/O SEL
2764
2716/32
+5
A12
A 7
A 6
A 5
A 4
A 3
A 2
A 1
A 0
D 0
D 1
D 2
GND
+5
+5
+5
A 8
A 9
+5/A11/MWR
MRD
A10
CE
D7
D6
D5
D4
D3
*BOARD SET UP FOR
2K CHIPS (#7 FOR RAM)

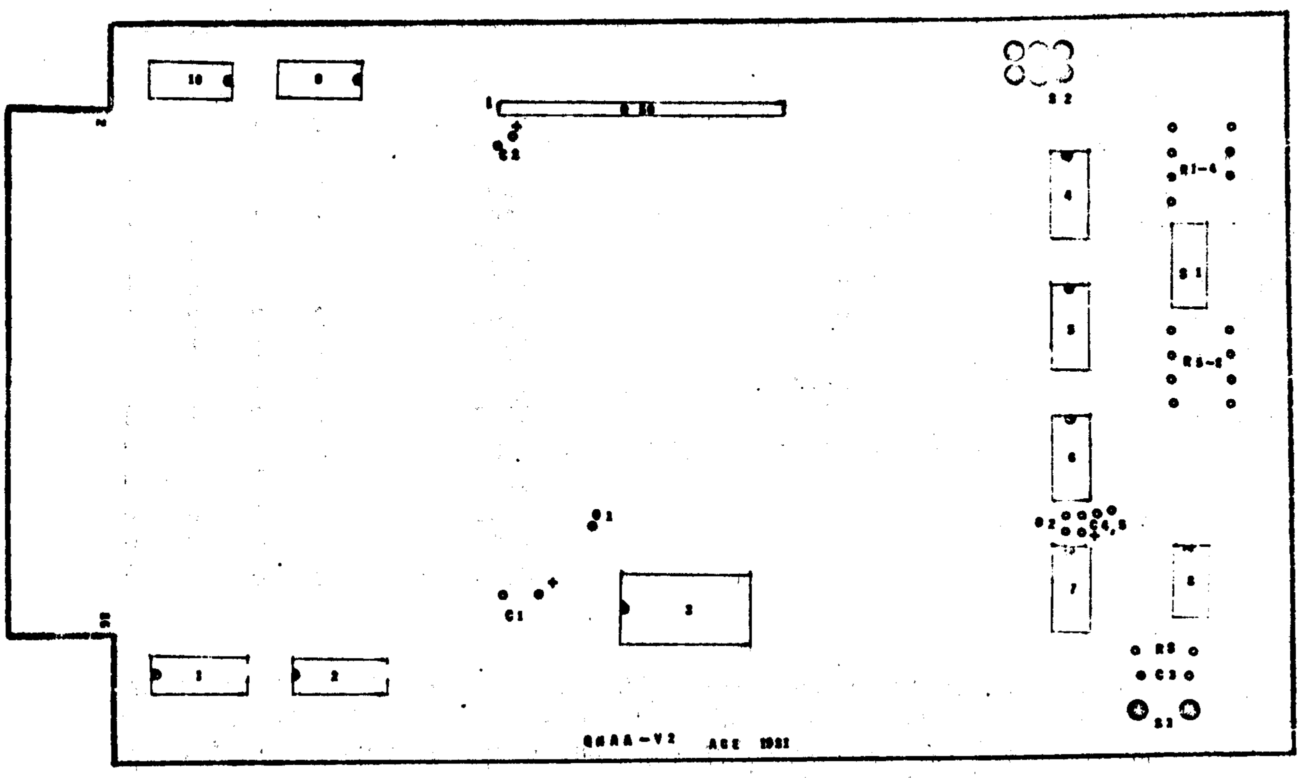
ACE QUEST - NETRONICS - ACE ADAPTER BOARD

Size - 5.0" x 11.0"

Function - a hard wire interface between the NAB 86 pin buss and the Quest 50 pin buss of the Super Expansion Board. In addition, 4k RAM and 4k EPROM (2716) switchable decoding, plus a separate fixed 2k EPROM monitor and 1k RAM stack and 1k Memory Mapped decoding are provided.

Power - ± 5 v., ± 12v., Gnd.

Documentation - Assembly instructions, Quest Super Expansion Board modification instructions.



QUEST NETRONICS ADAPTER BOARD (QNA)

Purpose: the QNA has been developed to provide a hardwired interface between the Netronics ELF II, via the Netronics adapter Board (NAB), and the Quest Super Expansion Board (SEB). In addition to generating the additional signals required, ie upper address, the QNA provides switch selectable 4k block decoding for the 4k of RAM and 4k of the EPROM (2716). The third Eprom socket is hardwired for address F000--F7FF, and the on board RAM, 2114s, is decoded at F800--FBFF as a remote stack. Memory map space is allowed at FC00--FFFF.

Switch S2 allows the 4k RAM and 4k Eprom to be interchanged to permit an EPROMED program to run at the RAM address, ie The TEXT EDITOR at 0000.

Note: some brands of EPROM will not work on the SEB, possibly due to gate delays, TI 2716 have been found to work.

Assembly: All parts except for the 50 pin header are located on top of the QNA. The board plugs into the upper 86 pin NAB socket, sitting behind the ELF II, and the SEB lies below and to the rear of the QNA. The edgeconnector of the QNA is extra long to permit adjustment of the placement of the two boards to suit your particular case configuration. Test the various positions prior to soldering the headers to ensure a suitable fit. It is suggested that the female header be placed on the QNA and the male on the SEB.

Parts:

2-	4050	IC 9,10	
2-	2114	1,2	
1-	4508	3	
3-	4585	4,5,6	
1-	74LS138	7	
1-	4070	8	
1-	8 position mini dip switch		S1
1-	DPDT		S2
8-	22k ¼w. resistors		R 1-8
1-	47k ¼w.		R 9
1-	0.01 uf. cap.		C 3
2-	10 uf. "		C1,2
2-	bypass caps.		C4,5
1-	pair male and female 50 pin header		Q50

Switch S1 selector table

address	S1	S2	S3	S4
0000	1	1	1	1
1000	0	1	1	1
2000	1	0	1	1
3000	0	0	1	1
4000	1	1	0	1
5000	0	1	0	1
6000	1	0	0	1
7000	0	0	0	1
8000	1	1	1	0
9000	0	1	1	0
A000	1	0	1	0
B000	0	0	1	0
C000	1	1	0	0
D000	0	1	0	0
E000	1	0	0	0
F000	0	0	0	0

Operation:

S1 provides a 4 bit pattern to the comparitors, IC 4 and 5, to select the decoder address of the 4k RAM and 4k EPROM on the SEB. A closed switch indicates a 1, and an open switch indicates a 0 in the above table. The decoding operates in the same manner as on the Netronics 4k memory boards.
S2 interchanges the decoder selector signal from the two comparitors to permit the RAM and the EPROM to replace each other (ie, move the EPROM from E000 to 0000, and move the RAM to E000).

S3 initiates a delay to the enable pin of the monitor EPROM, F000, as a boot function. Note: your monitor must be capable of utilizing a boot to work.

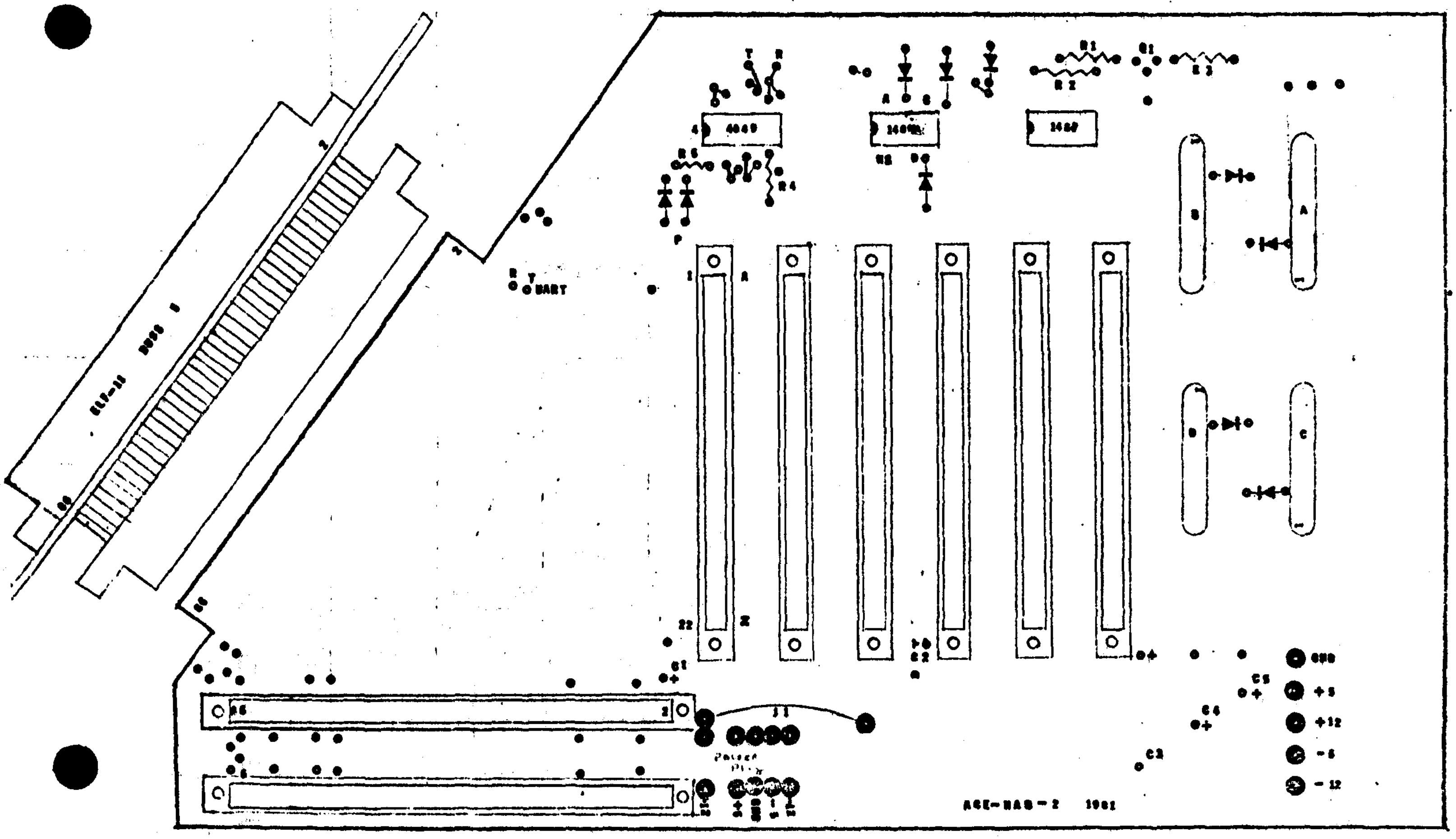
ACE NETRONICS - ACE - ADAPTER BOARD

Size - 7.0" x 12.0"

Funtion -a hard wire interface betwwen the 86 pin Netronics buss and the ACE 44 pin buss, and a TTL or RS-232C serial driver, receiver, and handshake circuit. Buss power distribution also available.

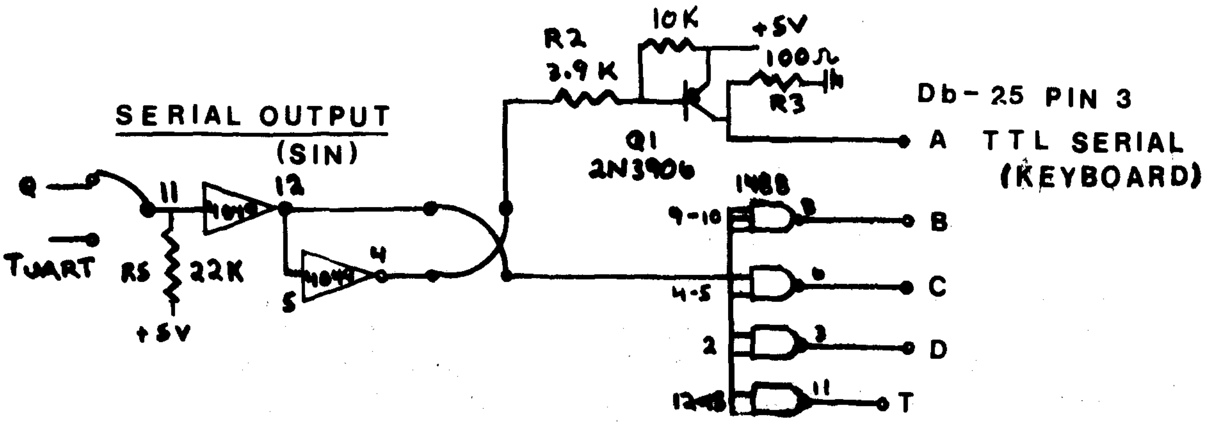
Power - ± 5 v., ± 12 v., Gnd.

Documentation - assembly guide, ELF 11 modification instructions.

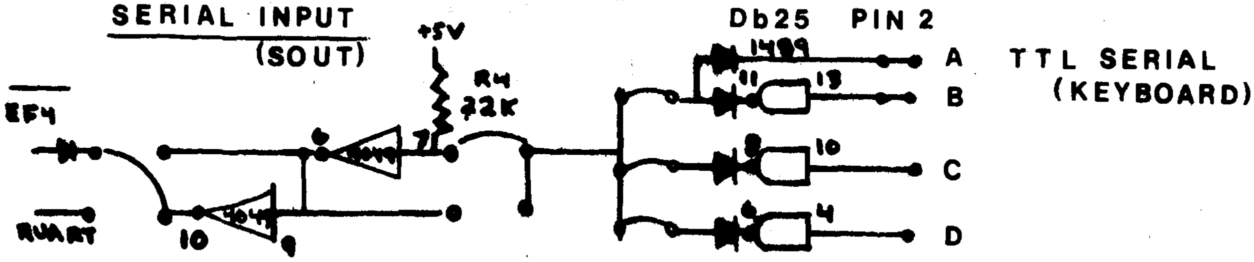


NETRONICS - ACE ADAPTER BOARD VER. 2, 1981

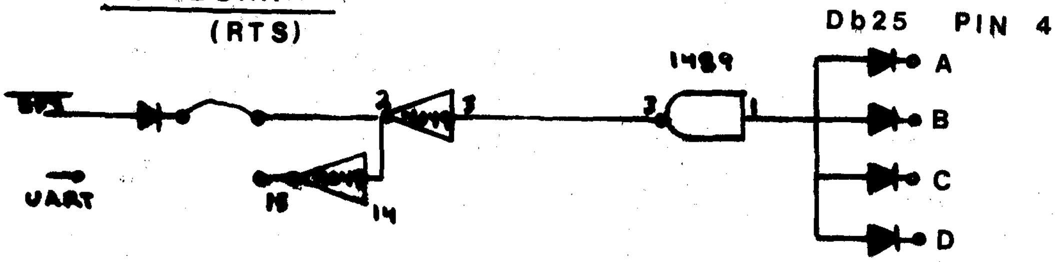
SERIAL OUTPUT (SIN)

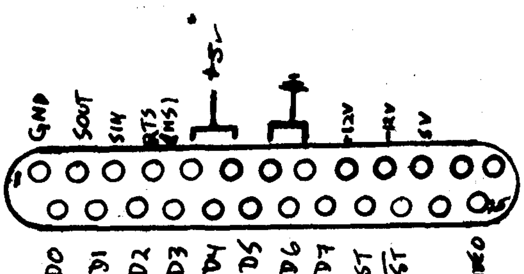


SERIAL INPUT (SOUT)



HANDSHAKE (RTS)

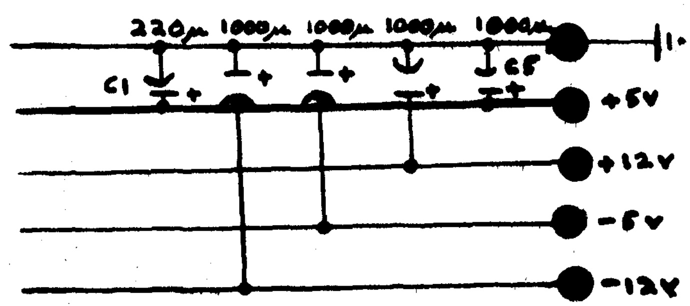




NOTE: Db25 CONNECTOR PIN OUT HAS BEEN MODIFIED TO PROVIDE POWER SUPPLY CONNECTIONS

* DOES NOT CONFORM TO RS232C CONFIGURATION

POWER SUPPLY



CLUB COMMUNIQUE

NAME________________________ DATE__________

PRODUCT ORDER	QUANTITY	UNIT PRICE	TOTAL
1 . Backplane and I/O board Ver. 2	______	$40.00	______
2. Front Panel W. EPROM Burner	______	June/82	______
3. I/O Adapter for Backplane Ver. 1	______	20.00	______
4. Kluge (wire wrap) Board	______	25.00	______
5. Netronics - Ace Adapter Board	______	25.00	______
6. Netronics - Quest Adapter Board	______	20.00	______
7. 8" Disk Controller Board	______	40.00	______
8. 64k Dynamic (4116) Board	______	50.00	______
9. DMA Adapter Board (ELF II)	______	3.00	______
10. EPROM (2716/32) Board	______	40.00	______
11.	______		______
12.	______		______

Software			
1. Fig Forth - Netronics Cassette (6k @0000H)	______	$10.00	______
2. Fig Forth - 2716 EPROM (6k @0000H)	______	30.00	______

Back Issues			
1. "Defacto" Year 1-3 (Edited)	______	$18.00	______
2. Year 4 Reprint	______	8.00	______

Membership			
Current year - Sept. 81 - Aug. 22, includes 6 issues of Ipso Facto	______	$18.00	______

VOTE FOR BEST ARTICLE

Issue________ Article______________________________

PRICE NOTE

Prices listed are in local funds. Americans pay in US funds. Canadians in Canadian funds. Overseas in US Funds. Overseas orders, for all items other than membership, add $2.00 US for air mail rather than parcel post. Please use money orders or bank draft for prompt shipment. Personal cheques require up to 6 weeks for bank clearance.

SALE POLICY

We guarantee that all our products work in an A.C.E. configuration micro computer. 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
