Ipso Facto

ISSUE 38 DECEMBER 83

INDEX PAGE

A PUBLICATION OF THE ASSOCIATION OF COMPUTER-CHIP EXPERIMETERS

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

1983-1984 EXECUTIVE OF THE ASSOCIATION OF COMPUTER CHIP EXPERIMENTERS

President:	John Norris	416-239-8567	**Vice-President:**	Tony Hill	416-876-4231
Treasurer:	Ken Bevis	416-277-2495	**Secretary:**	Fred Feaver	416-637-2513

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

Newsletter:

Production Manager:	Mike Franklin 416-878-0740	**Product Mailing:**	Ed Leslie 416-528-3222 (Publication)
Editors:	Fred Feaver Tony Hill		Fred Feaver 416-637-2513 (Boards)
Publication:	Dennis Mildon John Hanson		

Club Mailing Address:

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

ARTICLE SUBMISSIONS:

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

PUBLICATION POLICY:

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

MEMBERSHIP POLICY:

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

EDITORS CORNER

In answer to the anxious callers who queried the issue number on the last issue of Ipso Facto, it was in error. This is the second issue of the 1983/4 club year, #38.

This issue also contains something I didn't want to print -

BLATANT FILLER!!!!!

I have not received one article since September. The editor's closet is bare! The next issue of Ipso Facto may have only a front and rear cover, so, if YOU want to get more out of ACE then put something into it.

Club boards are selling well again. It is a curious cycle, with peaks that strip our supply. If you have ordered a front panel orbackplane recently, they are on order at the manufacfurers, so please be patient. The front panels are due within a week, the backplanes by the first week of January.

A new expanded version of CHIP 8 is on the way. With new commands, including an ASCII character set and better access routines, the old RCA games manuals can be put to use again.

Would members write in to me to express interest in the following boards : a revised disk board, with 5¼ and 8 inch drive capability.

: an 80 x 24 video board, based upon memory mapping the 6845

: a D to A and A to D board

: a modem

Your response will dictate whether the club invests in these new products.

Please feel free to write with other suggestions for boards.

MEMBERS CORNER

Carlos Qualls 1825 S. Ginger, Cornelius, Oregon 97113 USA
Projects I am currently working on and which I plan to write an article on as soon as possible. If you have any questions or would like to help then please write me.

1) Weather Station using an ELF to keep track of the time, temperature, wind speed etc. in my area. I am thinking about writing this in FORTH when I get it up and running.

2) RAM/ROM board using TMM6116 static ram chips which have the same pinouts as the 2716 ROM chip. I am looking at either a 16K or a 32K version.

3) Terminal Program with the capabilities of uploading and downloading code from a mainframe at 1200BAUD. This one is in the future as I will have to get a 1200BAUD modem and a second serial port on my ELF.

4) Small "C" V2 complier for the ELF. I have a copy of the Small "C" complier version 2 that was in DR DOBBS on a UNIX system I have access to. I plan to change the code to generate 1802 assembly language rather than the current 8080 assembly language. This will give me a "high level language" to write code for my 1802. DR DOBBS also has a UNIX-like OS written in small C v2 which might work well on the ELF.

5) RAM board using Intel's new 8K by 8 iRAM (integrated RAM). This is a dynamic RAM with refresh built onto the chip. These chips will allow a 64K board using less than 15 chips (8 memory chips and around 7 support chips).

QUESTIONS for the members.

Is there anyone out there who has access to USENET or ARPAnet or any other UNIX based network? If so I would like to talk to you over the network. My USENET address is omsvax!clq.

Does anyone have FORTH up and running on a cassette based system? Can the cassette be used to hold the blocks(screens) like the disk does?

Does anyone have the address to Netronics Tiny Basic I/O (terminal version)? I would like to substitute my own I/O routines to have Tiny Basic run at 1200 Baud on my system.

I would like the I/O address also for the Netronics Assembler. Text Editor and Disassembler? Again so I could substitute my own I/O routines to allow them to run at 1200 Baud on my system.

Netronics Tiny Basic I/O jumps are located at 0106 for input and 0109 for output. Long jumps are required.

Netronics Text Editor keyboard input is located at 0B79, and output is located at0CB2.

Perhaps someone out there could help with the other addresses.

Would anyone like to help me on the Small "C" to 1802 conversion? I will have it set up to generate I/O on my system and am not familiar enough with the ACE system to fix the I/O to correctly generate code for the ACE standard. For that matter exactly what is the ACE standard? I cannot remember it being talked about in any of the recent articles.

In relation to the above, has any member converted a Netronics ELF-II into an ACE standard machine? What are the differences? Is all I need an ACE to Netronics adapter board? How does the ACE machine do its I/O? Through a UART?

Thanks in advance and keep up the great work.
Carlos Qualls 1825 S. Ginger, Cornelius, Oregon USA 97113

FOR SALE:

Netronics Keyboard, video board and case assembled and working. Only needs a power supply and monitor(or TV) to be up and running again. Selling because I now own a terminal. MAKE OFFER.

Netronics 16K static RAM board assembled and working. Selling because I am building a 64K memory board. MAKE OFFER.

Carlos Qualls 1825 South Ginger, Cornelius, Oregon USA 97113

Homebrew ELF, 39k Ram, monitor on Rom,ACE Buss, ACE VDU board, power supply, 2 joy sticks, case. Cassettes of Chip 8 games, Tiny Basic, Quest Basic v5.0. Complete documentation.
$175.00
R. Nunnamaker, 111 Fairholt RD.S., Hamilton, Ont., L8M 2T6 416-547-9867.

9600 BAUD SERIAL I/O FOR 1802 4/21/83

G. JONES 7717 N. 46TH DR, GLENDALE, AZ 85301

The relentless up-grading process continues. As with most 1802 based home computers, my system started as a single board trainer, and as technology and personal finances allowed, hardware and softwate have been added, until my system is now nudging the limits of its memory capacity. A dot matrix printer, video monitor, full BASIC, and an editor/assembler are some of the major purchases I have made recently.

One of the first additions I made to my 4K Super Elf was a video monitor and ASCII terminal. Suddenly, I had graduated from Elf-Graphics block characters to a "real computer"! The 64x16 display generated by the Netronics VID-1 took me from the realm of the "toy" computer into the real world, and I began to realize the potential of my 1802 micro.

However, after acquiring an Okidata ML82A printer last December, I began to feel the need for an 80 column display. I could have added a memory mapped video board, but when Netronics announced the Smartvid-80 terminal, I found I could upgrade my present ASCII terminal by just replacing the VID-1 board with the Smartvid-80 board.

At first I continued to use the 300 baud I/O I had used with the VID-1, but the higher baud rates beckoned. I tried the higher speed switch settings with Super BASIC, which has a variable rate serial I/O written into it. The manual notes that "baud rates from less than 100 to approximately 4800 baud are automatically set by the software." I tried, but 2400 baud was the highest setting I could use. Super's baud rate timing counter decremented past zero, and I found myself back in the 10-100 baud range.

Then I remembered an article by Laylor Burdick that I had seen in the old Club 1802 newsletter, in which he implemented a switchable baud rate selector in order to use a H-9 terminal at 9600 baud, and a TTY-33 at 110 baud. A little careful dismembering of his routine, and the addition of a substitute for the Super BASIC startup, resulted in the following I/O routine which runs great on Quest's Editor/Assembler as well.

I was assembling my Smartvid board about the time M. Smith's review appeared in I.F. #32, but later I tried to simulate the problem he reported, about the screen going dead, and couldn't duplicate it. The one thing that aggravates me about the Smartvid-80 is its problem with the Lock Keyboard function. Sometimes my terminal will power up with the keyboard locked, or sometimes it will detect an "escape *" sequence in the data stream and lock up. There's no reset button, and the only recourse is to power the terminal off, or to have the host system send an Unlock Keyboard command (escape "). It's really annoying after you've just loaded 16K of program from cassette, only to find the keyboard is locked up, and you have to power down to reset it. Possibly, I could insert an Unlock Keyboard command in my boot-up header....someday.

```
0000            ;
0000            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
0000            ;
0000            ;    SERIAL 9600 BAUD I/O ROUTINE
0000            ;           WITH QUEST SUPER ELF OUTPUT VIA "Q" LINE
0000            ;           AND INPUT VIA SERIAL PORT ON EF2 FLAG LINE
0000            ;
0000            ;           SPECIAL INITIALIZATION INCLUDED FOR QUEST
0000            ;           SUPER BASIC.
0000            ;
0000            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
0000            ;
0000
3300                   ORG #3300
3300            ;
3300 C0330C INIT:  LBR S.INIT;    BRANCH VECTOR TO INITIALIZATION ROUTINE
3303 C03311 BREAK: LBR BREAK0;    BRANCH VECTOR TO BREAK ROUTINE
3306 C0341B OUTPUT:LBR OUT96;     BRANCH VECTOR TO OUTPUT ROUTINE
3309 C03400 INPUT: LBR IN96;      BRANCH VECTOR TO INPUT ROUTINE
330C            ;
330C            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
330C            ;
330C            ;    SUPER BASIC INITIALIZATION ROUTINE
330C            ;
330C            ;          REPLACES ENTIRE SUPER BASIC I/O INIT ROUTINE
330C            ;          WITH A SINGLE CLEAR SCREEN/FORM FEED COMMAND
330C            ;
330C            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
330C            ;
330C F80C   S.INIT:LDI #0C;       LOAD A CLS COMMAND INTO
330E BF            PHI RF;        RF.1
330F 3006          BR OUTPUT;     AND GO OUTPUT IT
3311            ;
```

```
3311        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
3311        ;
3311        ;    BREAK CHECK ROUTINE
3311        ;
3311        ;          CHECKS FOR ANY INPUT DURING THE OUTPUT
3311        ;          ROUTINES AND RETURNS WITH DF DET IF A
3311        ;          BREAK CONDITION EXISTS
3311        ;
3311        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
3311        ;
3311 FC00   BREAKO:ADI #00;     CLEAR DF
3313 3517          B2 BREXIT;   CHECK EF2 FOR BREAK CONDITION
3315 FF00          SMI #00;     YES, BREAK EXISTS, SET DF
3317 D5     BREXIT:RETN
3318        ;
3318        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
3318        ;
3318        ;    9600 BAUD SERIAL INPUT/OUTPUT ROUTINES
3318        ;         WITH ESPECIAL THANKS TO LAYLOR BURDICK
3318        ;
3318        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
3318        ;
3318
3400               ORG #3400
3400 FC00   IN96:  ADI #00;     CLEAR DF
3402 3502   WTST:  B2 WTST;     WAIT FOR A START BIT
3404 F880          LDI #80;     SET 8TH BIT TO A ONE
3406 C4     NXBIT: NOP;         NOP FOR TIMING
3407 3D0E          BN2 SPACE;   IF 1ST BIT IS A SPACE, GO THERE
3409 E2            SEX R2;      2 CYCLE NOP FOR TIMING
340A FC80          ADI #80;     NOT A SPACE, SO SET HIGH ORDER BIT
340C 3010          BR CONT;     AND BYPASS THE SPACE TIMING
340E C4     SPACE: NOP;         TIMING FOR A
340F C4            NOP;         SPACE BIT
3410 C4     CONT:  NOP;         MORE TIMING
3411 C4            NOP
3412 3317          BDF DONE;    IF ORIG. 8TH BIT IS SHIFTED INTO DF, DONE
3414 F6            SHR;         OTHERWISE, SHIFT RIGHT, END BIT INTO DF
3415 3006          BR NXBIT;    AND GO BACK BOR ANOTHER BIT
3417 C4     DONE:  NOP;         TIMING FOR
3418 C4            NOP;         THE FINAL
3419 C4            NOP;         (8TH) BIT
341A BF            PHI RF;      SAVE THE INPUT CHARACTER AND AND GO OUTPUT
341B        ;
341B        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
341B        ;
341B        ;    9600 BAUD OUTPUT ROUTINE
341B        ;
341B        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
341B        ;
341B 9F     OUT96: GHI RF;      GET THE OUTPUT CHARACTER
341C 7B            SEQ;         SEND A SPACE AS A START BIT
341D C4            NOP;         START BIT TIMING
341E FF00          SMI #00;     SET DF
3420 76            SHRC;        SHIFT DF INTO BIT 8, AND BIT 0 INTO DF
3421 C4            NOP
3422 C4     NXBITO:NOP;         MORE TIMING
```

```
3423 C4                NOP
3424 E2                SEX R2;      NOPS FOR 4 MACHINE
3425 E2                SEX R2;      CYCLES OF TIMING
3426 3B2A              BNF SP-OUT;  IF DF ISN'T SET, SEND A SPACE
3428 7A                REQ;         IF IT IS SET, SEND A MARK
3429 C8                LSKP;        AND DON'T
342A·7B        SP-OUT:SEQ;         SEND A SPACE
342B C4                NOP;         TIMING
342C 3231              BZ DUNOUT;   DONE IF ORIG. DF BIT HAS SHIFTED OUT
342E F6                SHR;         ELSE, SHIFT REMAINING BITS RIGHT
342F 3022              BR NXBITO;   AND GO BACK FOR ANOTHER BIT
3431 9F        DUNOUT:GHI RF;       NOW RESTORE THE ACCUMULATOR
3432 D5                RETN;        AND RETURN TO CALLING ROUTINE
3433
OBJECT ENDS AT:604A
:A#1

3300 C0330CC03311C0341BC03400F80CBF30
3310 06FC003517FF00D5
3400 FC003502F880C43D0EE2FC803010C4C4
3410 C4C43317F63006C4C4C4BF9F7BC4FF00
3420 76C4C4C4E2E23B2A7AC87BC43231F630
3430 229FD5
OBJECT ENDS AT:604A
```

ENHANCEMENTS TO HANNAN'S TEXT EDITOR

One of the first things a computer hacker wants to do when he gets a new addition to his computer system is to use it. I was no exception, so following the addition of my new Okidata printer to my Super Elf, I began to look for ways to utilize the new addition. Before long, I had listings of all my programs, and had experimented with the limited graphics of the printer. However, you can run just so many copies of a listing, and the time required to do anything useful with printer graphics can soon make that activity tedious, so I was intrigued when Fred Hannan's Text Editor appeared in Vol. 3, #2 of Questdata.

It wasn't long before I had the program on line, and found it to be quite useful to me. However, the program has several drawbacks which keep it from being a real "word processor". It was designed as a "line Editor", not a text editor, so you have to retype the entire line to correct a spelling mistake or change a word. Also, there is no way to add or delete a line of text, which I found to seriously inhibit my use of the program.

Here are some simple additions to the Text Editor which add an "Insert" and a "Delete" line command. They can be typed into the program with no other changes, and will make Mr. Hannan's simple utility much more useful.

```
324 IF S$="D" GOTO 1700
325 IF S$="d" GOTO 1700
328 IF S$="I" GOTO 1800
329 IF S$="i" GOTO 1800


1482 PRINT "DELETE LINE -    = D"
1484 PRINT "INSERT LINE -    = I"


1700 INPUT "DELETE WHICH LINE #"Q: IF Q=0 GOTO 210
1705 IF Q>A1 PRINT "LINE # DOESN'T EXIST.": GOTO 1700
1720 PRINT "DELETE LINE #";Q
1730 INPUT "Y OR N"Q$: IF Q$="Y" GOTO 1750
1740 IF Q$="y" GOTO 1750
1741 IF Q$="N" GOTO 1700
1742 IF Q$="n" GOTO 1700
1745 GOSUB 1610: GOTO 1700
1750 PRINT "LINE #";Q: PRINT A$(Q)
1755 FOR I=Q TO (A1-1):A$(I)=A$(I+1): NEXT I
1757 PRINT "LINE DELETED"
1760 A1=A1-1: GOTO 1700


1800 INPUT "INSERT NEW LINE # AFTER WHICH LINE #"Q: IF Q=0 GOTO 210
1810 IF Q>A1 PRINT "INVALID ENTRY": GOTO 1800
1820 A1=A1+1
1830 FOR I=A1 TO (Q+1) STEP -1
1840 A$(I)=A$(I-1): NEXT I
1845 PRINT "INPUT NEW LINE"
1850 INPUT Q$:A$(Q+1)=Q$: GOTO 1800
1900 END
```

CASSETTE TAPE REPAIRS

-by Dick Thornton 1403 Mormac Road, Richmond, Va. 23229

Cassette tape holders sometimes break. At times, the recorder may decide to eat a tape, ruining a section of it. A new tape may have a very long leader so that data is lost when writing. You may have a long, high quality tape, which could be better used as two reels, each containing half of the tape.

Correcting the above problems is simple if the holder is put together with screws. If not, you need to get one that is screwed together. Surplus houses sometimes offer these, and they can often be found in variety stores at low cost. I recently bought a pack of six cassettes for under $2. The tape was useless, but the holders were screwed together.

If the cassette to be modified is in a glued or heat-welded holder, carefully split the holder with a knife, saw, or whatever is handy, making sure you don't damage the tape. Inside, you will find two small plastic reels on which the tape is wrapped. The ends of the tape are held by snap-in pieces on the edge of the reels. Save the tape and reels and discard the rest.

Cassette holders with screws usually have 5 or 6 screws, one at each corner, one centered at the rear, and one centered near the front. Sometimes one or more screws may be under the paper label. Remove the screws, then gently separate the top and bottom of the holder, holding it horizontally, so parts don't fall out. If it wont separate easily, look for more screws. Study the arrangement of parts and how the tape is threaded. Draw a picture, if necessary. Discard the tape and reels if only the holder is to be used.

To free the end of the tape from a reel, start with the tape wound completely on the other reel. Press the small piece holding the tape out of the reel, which frees the tape. After cutting out the excess leader or bad section of tape, lay the reel on a table with the end of the tape lying against the cutout in the reel. Push one end of the holder piece into the cutout, capturing the tape, then snap the other end into the reel. Trim off the excess tape and you are through with this end. If the other end is to be modified, put the whole thing together and rewind the tape onto the other reel, then repeat the above for the other reel.

If you want to divide a long tape into two shorter pieces (for example, make two 30-minute cassettes from a 60-minute cassette) use your recorder to wind equal amounts of tape on each reel before taking the holder apart. In this case, you will want to save the reels from the cheap holder, as you will need two for each cassette.

STRUCTURED FLOWCHARTS
by Bob Briggs

Pretend you're a new programmer and you've just been asked to write a small program during a job interview. What's the first thing you show·your potential employer when you're ready with your program? A flowchart, right? This is what some instructors will tell you, anyway.

Without getting into whether or not you should use flowcharts (Why should you waste time on a flowchart --- you've got it all in your head! But you need it for documentation ... etc.) I'll describe one alternative to the traditional technique of diamonds and boxes and connecting lines that foster arbitrary transfers of control (i.e. GOTO's and hard to follow code.)

Nassi-Shneiderman (N-S) charts support structured programming concepts. After you have described your program with N-S charts, writing structured code follows easily, especially if you are using a structured language such as Pascal or the C programming language. (Is FORTH a structured language?). These charts were first publicized by Messrs. Nassi and Shneiderman in their article "Flowchart Techniques for Structured Programming," SIGPLAN notices of the ACM, v. 8, n. 8, Aug 1973. An article by C. Yoder and M. Schrag of IBM in Proceedings, ACM SIGSOFT/SIGMETRICS Software and Assurance Workshop, Nov 1978 (reprinted in "Tutorial on Software Design Techniques", by Freeman and Wasserman, 1980, IEEE Catalog No. EHO 161-0, available from IEEE Service Center, 445 Hoes lane, Piscataway, NJ, 08854) further describes and supports the use of these charts.

```
        START
          |
          |
         / \
       /     \    FALSE
     /  COND-  \___________
     \  ITION  /           |
       \  ?  /             |
         \ /               |
          |                |
          | TRUE           |
          |                |
      ---------        ---------
     |  "IF"   |      | "ELSE"  |
     | PROCESS |      | PROCESS |
      ---------        ---------
          |                |
          |<---------------'
          |
          |
         END
```

The IF-THEN-ELSE construct is represented using conventional flow charts as shown at the left.

The IF-THEN-ELSE construct is represented as follows using N-S charts:

```
 ------------------
|\                /|
| \  CONDITION   / |
|   \    ?     /   |
|     \      /     |
| TRUE  \  /  FALSE|
|------------------|
|         |        |
|  "IF"   | "ELSE" |
|         |        |
| PROCESS | PROCESS|
|         |        |
 ------------------
```

In both of the above charts, if the CONDITION is TRUE, the "IF" process is performed. If the condition is FALSE, the "ELSE" process is performed.

Similar N-S chart representations exist for other constructs such as WHILE, and DO UNTIL.

Here is an example of using the N-S charts. The problem is to read a keyboard to see if a key has been pressed, and to report only one keypress each time the typist presses a key. We will assume that the computer must continually query the keyboard to check for a key, and further assume that the computer does not have to handle the debounce problem --- i.e., the key only appears to go up and down once for each keystroke by the typist. Since the typist holds the key down for several queries by the computer (the computer is very fast), the computer has to keep track of when the key is down and when it is released.

To keep track of whether the computer has already reported the key we will use a variable, called FLAG. If FLAG is SET (TRUE), the key was already reported. If FLAG is RESET (FALSE), the key has been released or it has been pressed and the computer has not reported it yet.

Finally, if the ESCAPE key is pressed, the program will print "GOODBYE" and exit.

Here is the N-S chart:

```
:----------------------:
|   while (not ESC)    |
| :--------------------|
| |\                  /|
| |  \     key      /  |
| |    \   down   /    |
| |      \  ?   /      |
| |  yes   \  /    no  |
| |--------------------|
| |\  FLAG /|          |
| | \ set / |  reset   |
| |  \ ? /  |  FLAG    |
| |yes\ / no|          |
| |---------|          |
| |    |rept|          |
| |    |key |          |
| |    | &  |          |
| |    |set |          |
| |    |FLAG|          |
|----------------------|
|  print "GOODBYE"     |
|----------------------|
|       exit           |
 -----------------------
```

The "while(not ESC)" is my shorthand for "while the key pressed is not the ESCAPE key, continue with the contents of the while loop". The limit of the while loop is denoted by the extent of the vertical bar to the left of the while statement. In this example, it extends down to, but not including the print statement. By definition of the "while" statement, if "not ESC" is not true, (i.e. if the ESCAPE key is pressed), then program execution continues starting past the end of the while loop. This is the print statement in this case.

The body of the while loop contains two nested IF-THEN-ELSE constructs. If a key is not down, then the variable FLAG is reset (i.e. set to zero, or false, or not set). Then we have.reached the lower limit of the "while" loop and so go back to the top.

If a key is down, then FLAG is checked. If set, then the key was down the last time it was checked. Since we do not want to report it again, we do not do anything and go back to the top of the loop and check the "not ESC" condition again.

On the other hand, if a key is down and the flag is not set, then we do report the key and set the flag before going back to the top of the while loop.

As this example shows, it is very easy to follow and check the operation of the algorithm represented by the N-S chart. Control starts at the top and drops through one of several vertical channels depending on decision elements. Large programs are handled by breaking the code into segments small enough to fit onto one page.

I haven't drawn out the conventional flowchart for this example, but I believe it would be harder to follow, and more difficult to write structured code from. You're invited to try it and compare.

The articles mentioned above have more extensive examples of Nassi-Shneiderman charts. If their simplicity, readability, and codeability interest you, check them out!

16 BYTE WIDE HEX DUMP

A useful utility for the computer hacker is a hexadecimal memory dump program. In the back of the Quest Super BASIC V5.0 manual is a short hex dump by Ron Cenker. I tried using it, but found that the eight bit format was awkward, probably because I'm used to the output format that RCA used in their utility, UT4.

Here's a short Hex Dump program written in Super BASIC which will dump memory in 16 byte chunks, a la UT4.

Gary Jones
7717 N. 46th Drive
Glendale, Arizona 85301

DUMP EXAMPLE - @98D0 - @98FF

```
98D0: 8281 D1F8 01F1 FF06 4100 59F5 0047 4760
98E0: 800B 9881 0707 0000 820D 8080 3507 3C80
98F0: 0000 0000 0000 0000 0000 0000 0007 47AA
```

```
5 REM 16 BYTE HEX DUMP
15 REM G.L. JONES - QUEST SUPER BASIC V5.0
20 REM
25 DEFINT Z
30 DATA "0","1","2","3","4","5","6","7","8","9","A","B","C","D";"E","F"
35 FOR I=0 TO 15: READ H$(I): NEXT I
40 RESTORE
45 INPUT "ENTER START ADDRESS (@XXXX)"S1
50 INPUT "ENTER END ADDRESS (@YYYY)"E1
55 IF S1=E1 GOTO 45
60 CLS
65 GOSUB 400
70 S1=(S1/16)*16
75 FOR A=S1 TO E1 STEP 16
80 GOSUB 200: REM CALCULATE ADDRESS IN HEX
85 PRINT H$(A1)+H$(A2)+H$(A3)+H$(A4);": ";: REM PRINT ADDRESS
90 FOR W=0 TO 15 STEP 2
95 W1=PEEK(A+W):W2=PEEK(A+(W+1))
100 GOSUB 300: REM CALCULATE 4 DATA DIGITS IN HEX
105 PRINT H$(D1)+H$(D2)+H$(D3)+H$(D4);" ";
110 NEXT W: REM INCR WORD COUNT BY TWO
115 PRINT : NEXT A: REM INCR ADDRESS COUNT BY 16
120 PRINT CHR$(30): TOUT : INPUT "MORE"Q$
125 IF MID$(Q$,1,1)="Y" GOTO 45
130 END
200 A1=(A/4096):N1=A-(A1*4096)
205 A2=N1/256:N2=N1-(A2*256)
210 A3=N2/16:A4=N2-(A3*16)
215 RETURN
300 D1=W1/16:D2=W1-(D1*16)
305 D3=W2/16:D4=W2-(D3*16)
310 RETURN
400 INPUT "READY PRINTER"Q$
405 IF MID$(Q$,1,1)="" GOTO 415
410 IF MID$(Q$,1,1)="Y" GOTO 420
415 TOUT : RETURN
420 INPUT "PRINT SIZE - (S)MALL OR (R)EGULAR"Q$
425 IF MID$(Q$,1,1)="" GOTO 445
430 IF MID$(Q$,1,1)="R" GOTO 445
435 IF MID$(Q$,1,1)="S" GOTO 450
440 GOTO 420
445 POUT : PRINT CHR$(30): RETURN
450 POUT : PRINT CHR$(29): RETURN
```

WORD PROCESSOR II

INTRODUCTION

Having just finished my printer interface, I read (and loaded) with great enthusiasm the TEXT EDITOR in issue 24 of "Ipso Facto." While it answered a great need, it lacked a most important feature; a way for the user to edit a line, either increasing or decreasing the length without displaying the modification on final print out.

Having determined this need I started devouring all articles in past publications on word processors, including an article on the implementation of a word processor in North Star BASIC.[1]

After dissecting the listing it was apparent that a direct conversion was impossible; at least more difficult than writing one from scratch using the concepts presented. The following listing is the result of that effort.

PROGRAM SPECIFICS

Before implementing this program it is necessary to define the user area for the machine language programs, text storage area, and edit line buffer. As seen in the listing the program is configured for operation on a 36kword system. If this program is to be used with a different sized system, use the following memory map of the 36kword system in determing the user area.

MEMORY MAP

SUPERBASIC	0000-370F
iNPUT SUBROUTINE	3710-3737H
MOVE BACK SUBROUTINE	3738-373FH
MOVE FORWARD SUBROUTINE	3740-3757H
REPLACE SUBROUTINE	3758-3762H
TEXT CHARACTER BUFFER	3770-65FFH
EDIT LINE BUFFER	6600-66FFH
WORD PROCESSOR PROGRAM	6700-7BF8H
PROGRAM STACK AREA	7BF9-7EAFH

You will note that the program and stack area require ____H bytes of memory. It is suggested that to this you add an additional 512 bytes (for safety reasons). All other memory should be defined as user available (by using the DEFUS command).

Before loading the program, also modify the following lines:

a) line 50 - enter address of basic's input routine
b) line 100- enter address of basic's output routine
c) line 480- change arg3 to the length of text character buffer
d) line 500- change last argument to end of text character buffer address
e) line 560- change the 2E8F in arg3 to the length of the text character buffer
f) line 1840-change arg2 to the start location of the edit line buffer

g) line 1860-change both constants to start location of the edit line buffer

h) line 1870-change the 6600H to the start location of the edit line buffer

j) line 1920-change arg3 to the start location of the edit line buffer

Once all of the above changes are made to reflect the configuration of your system, define the user area and start entering the program.

Perhaps a word is in order about the machine language programs poked into memory at the start of the program. The first program allows the user to type faster than the same routine in BASIC would allow (this prevents those obnoxious program breaks). The next three programs are used purely to speed up the edit speed. Prior to writing these, a line edited at the start of a long text file would take about 5 minutes for the program to enter.

OPERATION

The word processor is fairly well prompting. The following functions are performed:

1. INPUT - input text to the text buffer
   - . NEW - start text at buffer's start
   - . CONTINUE - allow the user to continue adding text from the end of the file
2. PRINT - prints text to either the terminal or printer
   - . LINES NUMBERED? - if answered yes, the editor will print all lines with numbers
     - if answered no, the editor will automatically justify the right margin
3. EDIT - allow the user to modify text (on a line basis by entering shorter, equal, or longer lines
4. SAVE - saves the text buffer on tape
5. LOAD - loads the text buffer from tape
6. BYE - exits the word processor

***NOTE: The "^" are utilized to indicate line length (space indication). It is important to start and end the line the same as in the old listing so as not to mess up words in the adjacent lines.

To force carriage returns or indicate the start of a paragraph use the   key.

FUTURE MODIFICATION

The largest fault of this Word Processor is the lack of speed with which it justifies and prints text (48 words/minute). I suggest that someone (maybe me, if I get frustrated enough) write a machine language subroutine to replace lines 830 through 1300. This would greatly enhance the print speed.

A FINAL NOTE

I would be interested in hearing from anyone with comments/ modifications to this program and in addition will provide help in the implementation of this program if required. If a response is necessary please send a stamped, self-addressed envelope and allow for my lack of spare time when waiting for responses. My address is:

Tom Nery
33 County St.
Foxboro, Massachusetts 02035
USA

To give you an idea of the final results of the Word Processor II, this article was printed by it. Good luck to all who wish to copy it.

WORD PROCESSOR II

```
10 DEFINT Z
20 DIM Z(150)
30 REM INPUT SUBROUTINE
40 REM ***** ADDRESS OF THE INPUT ROUTINE *****
50 DATA #D4,#33,#09
60 REM ****************************************
70 DATA #FB,#08,#3A,#1A,#28,#30,#10,#9F,#FB,#0D
80 DATA #3A,#27,#F8,#0A
90 REM ***** ADDRESS OF THE OUTPUT ROUTINE*****
100 DATA #D4,#33,#06
110 REM ****************************************
120 DATA #F8,#20,#BF,#9F,#58,#18,#2A,#9A,#3A,#32
130 DATA #8A,#3A,#32,#D5,#9F,#FB,#04,#3A,#10,#D5
140 REM MOVE SUBROUTINE FOR NEW LINE < OLD LINE
150 DATA #48,#5A,#1A,#FB,#04,#3A,#38,#D5
160 REM MOVE SUBROUTINE FOR NEW LINE > OLD LINE
170 DATA #F8,#00,#AF,#BF,#1F,#48,#1A,#1F,#FB,#04,#3A
180 DATA #45,#08,#5A,#28,#2A,#2F,#9F,#3A,#4C,#8F
190 DATA #3A,#4C,#D5
200 REM SUBROUTINE TO REPLACE OLD LINE WITH NEW
210 DATA #0A,#FB,#04,#32,#62,#4A,#58,#18,#30,#58,#D5
220 FOR I=0 TO 82
230 READ A
240 POKE(@3710+I,A)
250 NEXT I
260 REM ENTER START OF TEXT MEMORY
270 E30=@3770
280 W=80
290 CLS
300 INPUT "INPUT, EDIT, PRINT, LOAD, SAVE, OR BYE"A$
310 IF MID$(A$,1,1)="I" GOTO 380
320 IF MID$(A$,1,1)="P" GOTO 580
330 IF MID$(A$,1,1)="E" GOTO 1540
340 IF MID$(A$,1,1)="L" GOSUB 2000: GOTO 290
350 IF MID$(A$,1,1)="S" GOSUB 1960: WAIT(100): GOTO 290
360 IF MID$(A$,1,1)="B" END
370 PRINT "PLEASE ANSWER I,E,P,L,S, OR B": GOTO 300
```

```
380 INPUT "IS THIS NEW OR CONTINUED INPUT"A$
390 IF MID$(A$,1,1)="N" GOTO 420
400 IF MID$(A$,1,1)="C" GOTO 500
410 PRINT "PLEASE ANSWER N OR C": GOTO 380
420 CLS: PRINT "READY TO ACCEPT TEXT INPUT"
430 REM *** THE FOLLOWING CALL IS TO THE MACHINE
440 REM *** LANGUAGE SUBROUTINE.  THE ARGUMENTS ARE:
450 REM ***     ARG 1 - SUBROUTINE ADDRESS
460 REM ***     ARG 2 - TEXT STACK STARTING LOCATION
470 REM ***     ARG 3 - MAXIMUM TEXT LENGTH
480 CALL (@3710,E30,@2E8F)
490 GOTO 290
500 FOR I=E30 TO @65FF
510 A=PEEK(I)
520 IF A=4 EXIT 550
530 NEXT I
540 PRINT "END OF TEXT NOT FOUND": GOTO 300
550 CLS: PRINT "READY TO ACCEPT TEXT INPUT CONTINUATION"
560 CALL (@3710,I,@2E8F-I-E30-2)
570 GOTO 290
580 REM *** THIS IS THE PRINT ROUTINE
590 CLS:C10=E30
600 INPUT "DO YOU WANT LINES NUMBERED"A$
610 IF MID$(A$,1,1)="Y" GOTO 760
620 IF MID$(A$,1,1)="N" GOTO 640
630 PRINT "PLEASE ANSWER Y OR N": GOTO 600
640 GOSUB 2040
650 PRINT : INPUT "HOW MANY LINES PER PAGE"P:P=P+1
660 PRINT : INPUT "WHAT IS STARTING LINE NUMBER"P1
670 PRINT : INPUT "PRINTER OR TERMINAL OUTPUT DEVICE"O$
680 O$=MID$(O$,1,1)
690 IF O$<>"P" IF O$<>"T" PRINT "PLEASE ANSWER P OR T": GOTO 670
700 L5=1: IF O$="T" CLS: GOTO 830
710 PRINT : INPUT "SINGLE OR DOUBLE SPACED"S$
720 S$=MID$(S$,1,1)
730 IF S$<>"S" IF S$<>"D" PRINT "PLEASE ANSWER S OR D": GOTO 710
740 IF S$="D"L5=2
750 GOTO 830
760 PRINT : INPUT "LINE NUMBER RANGE (LOW,HIGH)"N8,N9
770 GOSUB 2040
780 PRINT : INPUT "PRINTER OR TERMINAL OUTPUT DEVICE"O$
790 O$=MID$(O$,1,1)
800 IF O$<>"P" IF O$<>"T" PRINT "PLEASE ANSWER P OR T": GOTO 780
810 L5=1
820 J9=0: GOTO 1310
830 IF P1=1 IF O$="P" POUT : PRINT : PRINT : PRINT : PRINT : PRINT :P1=P1+10
840 J=E30
850 K=0
860 W=W+1
870 FOR I=J TO J+W
880 K=K+1
890 Z(K)=PEEK(I)
900 NEXT I
910 REM *** SEARCH FOR NUMBER OF SPACES AND PARAGRAPH
920 REM *** DELIMITER (@)
930 S=0:S10=0
940 M=W
```

```
950 N=1
960 FOR I=1 TO M
970 IF Z(I)<>#20 IF Z(I)<>#40 THENS10=1
980 IF Z(I)=#20 IF S10=1 THENS=S+1:S1=I+1-N
990 IF Z(I)=#40 IF I<>1I=I-1: EXIT 1180
1000 IF Z(I)=#04 EXIT 1240
1010 IF I=1 IF Z(I)=#40 THENM=M+1:N=N+1
1020 NEXT I
1030 K=W-S1:S10=0
1040 IF Z(S1-1)=#20K=K+1
1050 FOR L=N TO S1
1060 IF Z(L)<>#20S10=1
1070 PRINT CHR$(Z(L));
1080 IF K<>0 IF Z(L)=#20 IF S10=1 PRINT " ";:K=K-1
1090 NEXT L
1100 PRINT :J=J+L-1:K=0:P1=P1+L5
1110 IF L5=2 PRINT
1120 IF P1<P GOTO 870
1130 P1=11: IF O$="P" PRINT : PRINT : PRINT : PRINT : PRINT
1140 TOUT : CLS: INPUT "PRESS <CR> TO CONTINUE"A$: CLS
1150 IF O$<>"P" CLS: GOTO 870
1160 POUT : PRINT : PRINT : PRINT : PRINT : PRINT
1170 GOTO 870
1180 FOR L=N TO I
1190 IF Z(L)=#40 GOTO 1210
1200 PRINT CHR$(Z(L));
1210 NEXT L
1220 J=J+1
1230 GOTO 1100
1240 FOR J=N TO I
1250 IF Z(J)=#40 GOTO 1280
1260 IF Z(J)=#04 GOTO 1280
1270 PRINT CHR$(Z(J));
1280 NEXT J
1290 PRINT
1300 TOUT :W=W-1: GOTO 290
1310 IF O$="P" GOTO 1330
1320 GOTO 1340
1330 POUT
1340 W=W-8
1350 FOR I=N8 TO N9
1360 GOSUB 1480
1370 W10=W*(I-1)
1380 FOR J=W10 TO W10+W-1
1390 A=PEEK(J+E30): IF A=#04 EXIT 1420
1400 PRINT CHR$(A);
1410 NEXT J
1420 PRINT : IF L5=2 PRINT
1430 IF A=#04 EXIT 1450
1440 NEXT I
1450 TOUT :W=W+8
1460 INPUT "PRESS <CR> TO CONTINUE"A$
1470 GOTO 290
1480 IF I<10J=3: GOTO 1520
1490 IF I<100J=2: GOTO 1520
1500 IF I<1000J=1: GOTO 1520
1510 J=0
1520 PRINT TAB(J);I;TAB(8);
1530 RETURN
```

```
1540 CLS: PRINT "THE EDITOR IS BASED ON ";W-8;" CHARACTER LINES"
1550 INPUT "DO YOU WISH TO CHANGE IT"A$
1560 IF MID$(A$,1,1)="Y" GOTO 1590
1570 IF MID$(A$,1,1)="N" GOTO 1610
1580 PRINT "PLEASE ANSWER Y OR N": GOTO 1550
1590 PRINT "REMEMBER, EDIT LINE LENGTH = LENGTH-8"
1600 INPUT "ENTER LINE LENGTH"W: GOTO 1540
1610 W=W-8
1620 CLS: PRINT "ENTER LINE TO BE EDITED (0 TO EXIT)"
1630 INPUT "LINE NUMBER"E5
1640 IF E5=0W=W+8: GOTO 290
1650 PRINT : PRINT
1660 W10=W*(E5-1)
1670 I=E5
1680 GOSUB 1480
1690 FOR J=W10 TO W10+W-1
1700 A=PEEK(J+E30)
1710 IF A=#04 EXIT 1740
1720 PRINT CHR$(A);
1730 NEXT J
1740 PRINT : PRINT TAB(8);
1750 FOR J=1 TO W
1760 PRINT "^";
1770 NEXT J
1780 PRINT : PRINT
1790 INPUT "DO YOU WISH TO EDIT THIS LINE"A$
1800 IF MID$(A$,1,1)="Y" GOTO 1820
1810 GOTO 1620
1820 PRINT "ENTER LINE (250 CHAR. MAX), CTRL-D TO END"
1830 PRINT
1840 CALL (@3710,@6600,@00FF)
1850 CLS: PRINT "PLEASE WAIT FOR THE PROMPT"
1860 FOR I=@6600 TO @6600+#FF
1870 IF PEEK(I)=#04I=I-@6600: EXIT 1890
1880 NEXT I
1890 IF I=W GOTO 1920
1900 IF I>W GOTO 1940
1910 CALL (@3738,W10+W+E30,W10+I+E30)
1920 CALL (@3758,W10+E30,@6600)
1930 GOTO 1620
1940 CALL (@3740,W10+W+E30,W10+I+E30)
1950 GOTO 1920
1960 REM SAVE TEXT SUBROUTINE
1970 INPUT "POSITION TAPE AND PRESS <CR> TO START"A$
1980 PSAVE C
1990 RETURN
2000 REM LOAD TEXT SUBROUTINE
2010 INPUT "POSITION TAPE AND PRESS <CR> TO START"A$
2020 PLOAD C
2030 RETURN
2040 PRINT : PRINT "WHAT LINE LENGTH (CURRENTLY SET AT ";W;
2050 INPUT " )"W
2060 RETURN
```

8/14/83

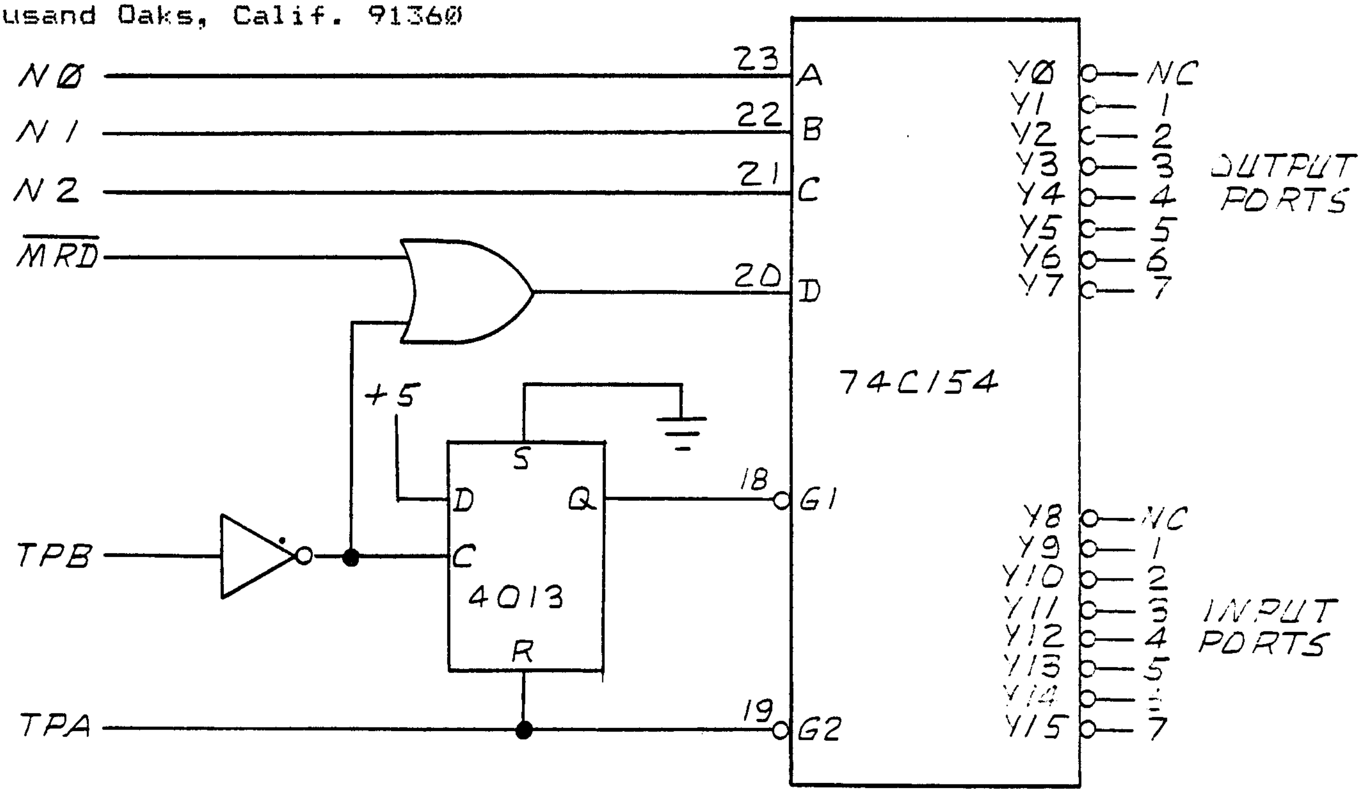
Association of Computer Experimenters ..
c/o M.E. Franklin
69Ø Laurier Ave.
Milton. Ontario
Canada L9T 4R5

Okay, okay, so you want articles for the newsletter. I have been meaning to send this one for a while, so here it is.

When adding I/O ports to the 18Ø2, extra circuitry must be added to gate the various control signals, ie MRD, TPA, MWR to each port. I have designed a way to gate these signals into the decoder so that no additional gating is required. All 14 ports are encoded, and all outputs are active low, which is what is needed in most cases. All that is needed is one 74C154, one half of a 4Ø13 D type flip-flop, one inverter, and one two input NOR gate.

Note that the YØ and Y8 outputs are not used.

Richard M. Cox
267Ø Calle Abedul
Thousand Oaks, Calif. 9136Ø



SHORT MEMORY TEST PROGRAM

THIS MEMORY TEST PROGRAM IS RELOCATABLE & REQUIRES LESS THAN ONE PAGE OF MEMORY. IT LOCATES THE TOP OF A CONTINUOUS BLOCK THEN PATTERNS EACH PAGE, TESTS THE PAGE FOR ERRORS THEN SHIFTS THE PATTERN RIGHT ONE BYTE. EACH BYTE IS TESTED 256 TIMES AND STOPS ON ERRORS. ANY KEY PRESS CONTINUES. HERE IS A LIST OF MONITOR CALLS USED :

D4 E0 07 XX	OUTPUT INLINE BYTE
D4 E0 0E	OUTPUT 16 BITS OF RA AS HEX EXPRESSION
D4 E0 14	INPUT, WAIT FOR IT
D4 E0 11	OUTPUT AN ASCII STRING, TERMINATE WITH MSB SET
C0 E0 00	LBR TO MONITOR
F8 01 BE	LOAD FIRST PAGE OF TEST BLOCK

IN MY SYSTEM, 20K TAKES 7.5 MINUTES WITH 3.58MHZ CLOCK.
BEWARE SYSTEMS WITH 64K CONTINUOUS RAM.

LYNN KEENLISIDE
LONDON, ONT.

```
0000   F8 01 BE BD F8 00 AE AD  EE 9E FC 01 BE 0E FB FF  x.>=x..-n.|.>.{
0010   5E F3 32 09 D4 E0 11 0C  54 45 53 54 49 4E 47 20  ^s2.T`..TESTING
0020   4D 45 4D 4F 52 59 A0 8D  AA 9D BA D4 E0 0E D4 E0  MEMORY .x.:T`.T`
0030   11 20 54 4F A0 8E AA 9E  BA 2A D4 E0 0E D4 E0 11  . TO .x.:xT`.T`.
0040   0D 0D 8D 8D AA 9D BA D4  E0 0E D4 E0 07 0D F8 00  ....x.:T`.T`..x.
0050   A9 8D 52 E2 89 5D 1D 19  8D 3A 60 9D FF 01 BD 8D  ).Rb.]...:`..=.
0060   F3 3A 54 F8 00 A9 8D 52  ED 89 F3 32 84 D4 E0 11  s:Tx.).Rm.s2.T`.
0070   0D 45 52 52 4F 52 20 41  54 A0 9D BA 8D AA D4 E0  .ERROR AT .:.xT`
0080   0E D4 E0 14 1D 19 E2 8D  3A 8F 9D FF 01 BD 8D F3  .T`...b.:...=.s
0090   3A 68 8D FC 01 AD 3A 4E  D4 E0 07 0B 9D FC 01 BD  :h.|.-:NT`...|.=
00A0   52 E2 9E F3 3A 43 D4 E0  11 0D 54 4F 50 20 4F 46  Rb.s:CT`..TOP OF
00B0   20 42 4C 4F 43 4B 20 20  4C 4F 4F 50 20 3F A0 6F   BLOCK  LOOP ? o
00C0   FA 7F FB 59 32 00 FB 17  C2 E0 00 30 BF 00 00 00  z{Y2.{.B`.0?...
00D0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
00E0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
00F0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
```

FORTH : Right 1802 Assembly Code

By- David Horner 15 Sadlee Cove Cr Agincourt Ont M1V 1Y3

I am a newcomer to 1802. Actually, I've had the chip sitting on my shelf for 8 yrs and recently undertook to build a computer system for my son. I saw FORTH as an ideal vehicle to drive this bare bones system. However, I was bothered by it's interrupt problems as my design uses interrupts heavily.

As mentioned by Tony Hill in the last newsletter +LOOP and LOOP contain problem code. However, these are readily fixed in place. The fix for the word "I" is a real challenge as the machine code could not be contained in the space required necessitating patching code in some "available space". This would generate non-standard variations of the club FORTH which I did not consider acceptable. Therefore, I sought a solution that would fit in the original space. What, as we all know, uses less memory than assembler code? FORTH of course!

The word "I" gets the current value from the return stack and can be defined as follows

```
: I >R DUP >R ; (10 bytes)
      or more simply
    : I R ; (6 bytes)
```

Which indicates "I" is the same as R! The code for R is identical to the code proposed for "I". "I" can therefore be implimented by changing "I"'s pointer to point to R. Changing the pointer allows for a standard solution and is faster and smaller than any patch. LOOP , +LOOP and "I" can be fixed as follows

```
     LOOP              +LOOP            I

00EE 92 B8 B7     0126 92 B8 B7    1423 03 BF
     82 A8 A7          82 A8 A7
     18 17             18 17
```

Locations 1425 - 1430 are available for patching.

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

HOORAY! A CLUB STANDARD MONITOR AT LAST. I OBTAINED A LISTING OF "SYMON CSC" AT THE APRIL CLUB MEETING AND SPENT SEVERAL HOURS LOADING IT IN MY ELF II WITH NETRONICS VID. THE FOLLOWING PATCHES WERE MADE TO "SYMON" TO ENHANCE ITS OPERATION WITH MY SYSTEM.

FIRST, I DO NOT HAVE A REAL TIME CLOCK, SO THE BYTES AT C03DH WERE CHANGED FROM D4 C7 8C TO C4 C4 C4. THIS KEEPS JUNK FROM PRINTING ON THE SCREEN EVERYTIME THE MONITOR IS ENTERED.

SECOND, SINCE I OFTEN HAD A SCREEN FULL OF GARBAGE WHEN ENTERING THE MONITOR, I WANTED TO DO A FF UPON "SYMON" INITIALIZATION. CHANGE C036H-C03CH FROM C4 C4 C4 D4 C1 C1 18 TO D4 C1 C1 18 D4 C1 78. THIS IS DONE AT THE COST OF 3 FREE BYTES IN THE INITIALIZATION CODE, BUT STILL LEAVES 3 BYTES FREE TO JUMP TO ANY INITIALIZATION CODE REQUIRED BY YOUR SYSTEM. THIS IS EASILY DONE USING THE SCRT BUILT INTO "SYMON".

THIRD, SINCE MY VID DOES NOT HAVE A HANDSHAKE LINE, IT IS NECESSARY TO DO A DELAY WHILE PERFORMING A FF. IF THIS DELAY IS NOT USED 2 OR 3 CHARACTERS WILL BE LOST AT THE START OF THE DISPLAY. FIRST, LOAD THE FOLLOWING CODE IN FREE SPACE WITHIN THE MONITOR.

```
C414  D4 C1 86        OUTPUT FF
CR17  0C 00
C419  D4 C1 DD        SAVE REG 8-B
C41C  F8 40 BB AB     DELAY COUNT
C420  2B 9B
C422  3A 20           LOOP UNTIL DONE
C424  D4 C1 EE        RESTORE REG 8-B
C425  D5
```

THEN CHANGE C178H-C17CH FROM D4 C1 86 0C 00 TO D4 C4 14 C4 C4.

FOURTH, WHEN I ENTERED AN ILLEGAL COMMAND, "SYMON" CRASHED. CHANGE C011H FROM 3A TO 42.

FIFTH, THE BAUD RATE IS SET AT 1200. THE "SYMON 3" LISTING IN IF#30 INCLUDED THE FOLLOWING TABLE.

BAUD	1/2 DUPLEX	FULL
150	49	48
300	25	24
600	13	12
1200	09	08

LOAD THE PROPER VALUE FOR YOUR SYSTEM IN C702H. I FOUND A VALUE OF 26H WAS NEEDED FOR MY 300 BAUD NETRONICS VID TO PROPERLY ECHO KEYBOARD INPUT CHARACTERS, BUT 25H WORKED FINE FOR HALF DUPLEX.

I AM CURRENTLY RUNNING "SYMON" OUT OF RAM UNTIL I AM SATISFIED WITH THE WAY IT RUNS IN MY SYSTEM. I HAVE INCLUDED ALL ADDRESSES OF CHANGED LOCATIONS AND THEIR PREVIOUS CONTENTS BECAUSE I AM NOT SURE MY LISTING IS THE SAME AS THE DISTRIBUTED LISTING.

AS FOR "SYMON", I LOVE THE DISASSEMBLER, BUT MISS THE REGISTER SAVE AND RESTORE OF "SYSMON". ALSO, THE I/O NEEDS WORK, BUT I WILL TAKE CARE OF THIS WHEN I COMPLETE THE CLUB CPU BOARD WITH THE HARDWARE UART. THANK'S AGAIN MIKE.

```
10 REM***PEPSI BOTTLE TOP CONTEST***
20 CLS
30 PRINT TAB(20);"PEPSI BOTTLE TOP GAME": PRINT : PRINT
40 INPUT "DATA FROM KEYBOARD OR TAPE"I$
50 IF MID$(I$,1,2)="KE" GOTO 770
60 IF MID$(I$,1,2)<>"TA" PRINT "CAN'T UNDERSTAND": WAIT(300): G
OTO 20
70 CLS: PRINT "PLACE DATA TAPE IN RECORDER": PRINT
80 INPUT "PUT RECORDER IN PLAY MODE AND PRESS RETURN"I$
90 DLOAD C,1,1: CLS
100 PRINT "DO YOU WANT TO:": PRINT
110 PRINT TAB(10);"ENTER NEW DATA"
120 PRINT TAB(10);"CHECK FOR WINNING NUMBERS"
130 PRINT TAB(10);"LIST NUMBERS"
140 PRINT TAB(10);"SAVE DATA ON TAPE"
150 PRINT TAB(10);"END PROGRAM"
160 INPUT "ENTER ONE OF THE ABOVE"I$
170 IF MID$(I$,1,3)="ENT" GOTO 230
180 IF MID$(I$,1,3)="CHE" GOTO 460
190 IF MID$(I$,1,3)="LIS" GOTO 550
200 IF MID$(I$,1,3)="SAV" GOTO 730
210 IF MID$(I$,1,3)="END" CLS: END
220 PRINT "CAN'T UNDERSTAND": WAIT(300): CLS: GOTO 100
230 CLS: PRINT "DO YOU WANT TO:": PRINT
240 PRINT TAB(10);"ENTER A TOP NUMBER"
250 PRINT TAB(10);"ENTER A WINNING NUMBER"
260 PRINT TAB(10);"RETURN TO MENU"
270 INPUT "ENTER ONE OF THE ABOVE"I$
280 IF MID$(I$,1,2)="TO" GOTO 320
290 IF MID$(I$,1,2)="WI" GOTO 390
300 IF MID$(I$,1,2)="ME" CLS: GOTO 100
310 PRINT "CAN'T UNDERSTAND": WAIT(300): CLS: GOTO 230
320 CLS: PRINT "ENTER PEPSI BOTTLE TOP NUMBERS"
330 PRINT "TO END INPUT ENTER 0 (ZERO)": PRINT
340 FOR A=T1 TO 250
350 INPUT "TOP # = "T(A)
360 IF T(A)=0 LET T1=A: EXIT 380
370 NEXT A
380 GOTO 230
390 CLS: PRINT "ENTER PEPSI WINNING NUMBERS"
400 PRINT "TO END INPUT ENTER 0 (ZERO)": PRINT
410 FOR B=W1 TO 30
420 INPUT "WINNING # = "W(B)
430 IF W(B)=0 LET W1=B: EXIT 450
440 NEXT B
450 GOTO 230
460 CLS: PRINT "CHECK FOR WINNING COMBINATION": PRINT
470 PRINT "WAIT A MINUTE WHILE I CHECK": PRINT
480 FOR B=1 TO W1-1
490 FOR A=1 TO T1-1
500 IF W(B)=T(A) PRINT TAB(10);"I FOUND ONE ";W(B);"+";T(A)
510 NEXT A
520 NEXT B
530 INPUT "END OF CHECK - PRESS RETURN TO CONTINUE"I$
540 CLS: GOTO 100
```

```
550 CLS: PRINT "LIST NUMBERS IN MEMORY": PRINT
560 PRINT "BOTTLE TOP NUMBERS:"
570 LET C=0
580 FOR A=1 TO T1-1
590 PRINT TAB(10*C);T(A);:C=C+1
600 IF C>4 LET C=0: PRINT " "
610 NEXT A
620 PRINT
630 INPUT "PRESS RETURN TO CONTINUE"I$: CLS
640 PRINT "WINNING NUMBERS:": PRINT
650 LET C=0
660 FOR A=1 TO W1-1
670 PRINT TAB(10*C);W(A);:C=C+1
680 IF C>4 LET C=0: PRINT " "
690 NEXT A
700 PRINT
710 INPUT "PRESS RETURN TO CONTINUE"I$: CLS
720 GOTO 100
730 CLS: INPUT "PLACE RECORDER IN RECORD MODE AND PRESS RETURN"
I$
740 DSAVE C,1,2
750 PRINT "DATA SAVED ON TAPE": WAIT(500): CLS
760 GOTO 100
770 LET T1=1: LET W1=1
780 DIM T(250),W(30)
790 GOTO 100
800 END
```

THE 8 BIT OUTPUT SCAM REVEALED

As we all know, the 1802 has seven (not including memory mapped I/O), 8 bit output ports, right? WRONG!!! The 1802 does in fact have seven output ports but, (its not your fault RCA has mislead us in all their literature) each output port can have up to 16 bits. This effectively doubles its output capability. Before going any further, let's look at how the 1802 does an output.

Once an output instruction is recieved (as we instructed it) it sets its mrd line low, puts the high byte address of the X register onto the address bus, strobes TPA, puts the low byte address of the X register onto the address bus, and then on the next clock cycle reads (once again, READS) that memory address. At the same time that the mrd line goes low, the N-lines go to their given state as determined by the second nibble of the output instruction. They stay in this state as long as the mrd line stay low.

You have probably figured out how to implement the 16 bit outputs after the last paragraph. By using the circuit shown in figure 1, we can use the mrd, TPA and the decoded N-line as the strobe to a pair of 4-bit latches for the high order byte and then use the same circuit except replace TPA with TPB for the low order byte.

To use this new output port, all that is required is to load the two bytes to be output into a register, set it to the X register and then perform the appropriate output instruction (the one which selects that port).

This circuit can be a great benefit in a small dedicated controller where the logic required for memory mapped I/O is a large part of the circuit. Let me assure you that the circuit does work as I have implemented it in various forms for some simple control applications.

Tom Nery, 33 County St., Foxboro, Massachusetts,USA,02035

ACE CPU Board

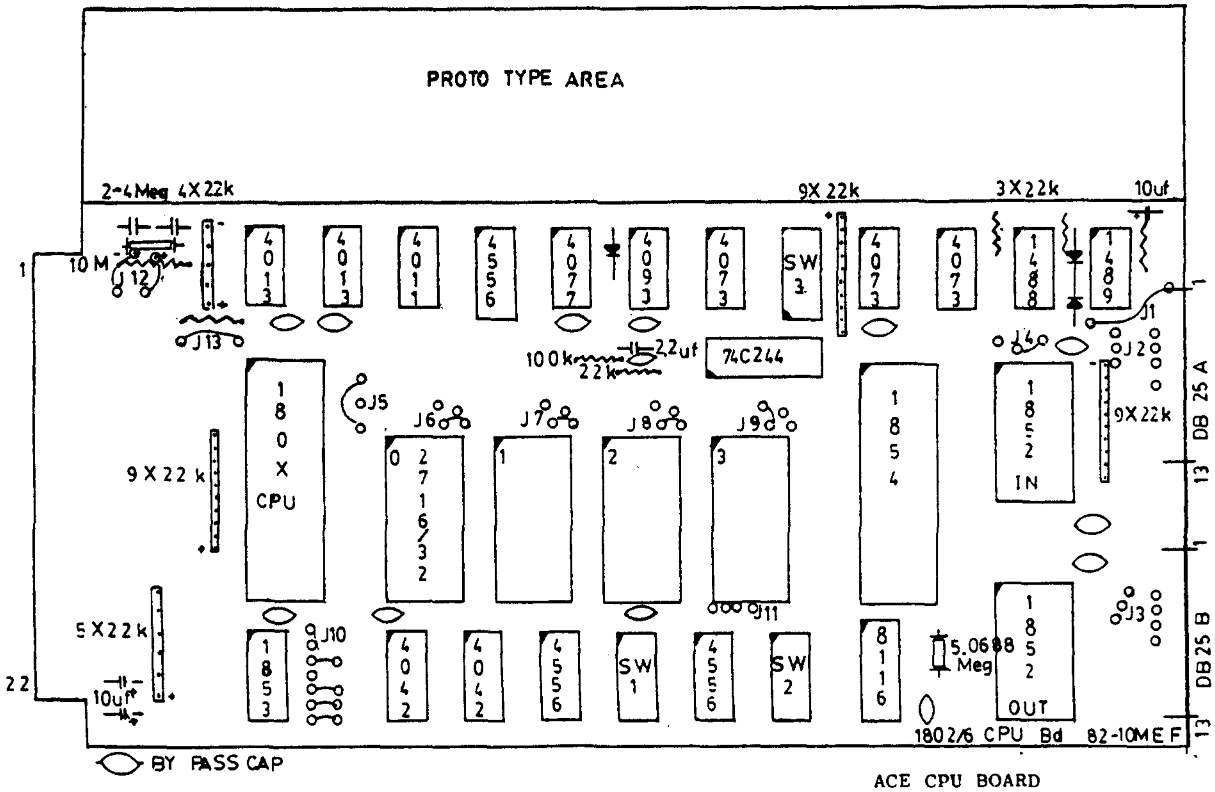
Size: 6" x 9.5"

Function: to provide - a system micro computer (1802 -04 -05 -06)
- control logic, power on reset, fully decoded INTERUPT, DMAIN and DMAOUT.
- selectable BOOT to any PAGE ADDRESS
- 4 JEDEC EPROM/RAM sockets, with DUAL ADDRESS decoding for 2 locations or sizes of memory.
- INPORT and OUTPORT
- UART with selectable baud rate
- RS 232 C with 2 Db 25 connectors
- extensive prototype area (1.5" x 8.5")

CPU Board is designed to be a system or standalone micro controller board.

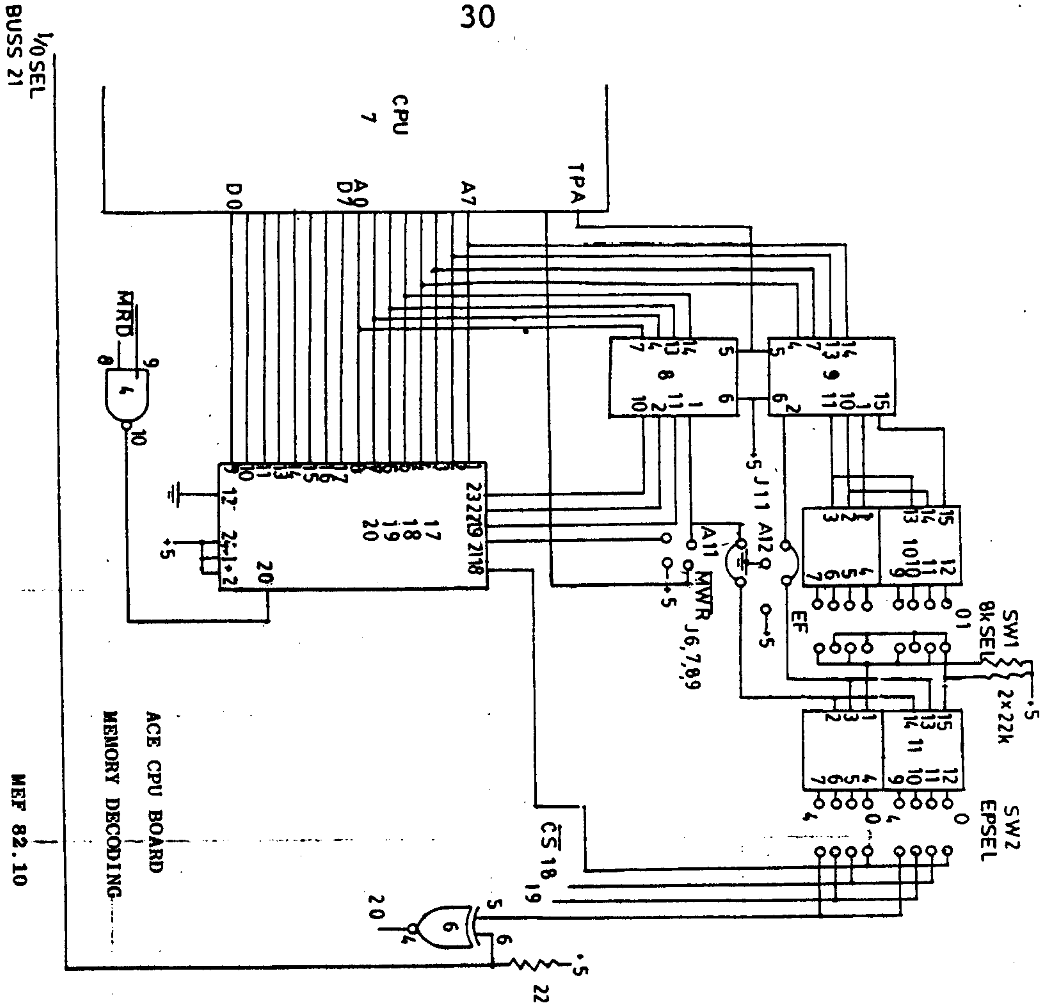
Power: +5 v. Gnd. ±12v for RS 232C circuit.

Documemtation: assembly and test instructions, software for UART.

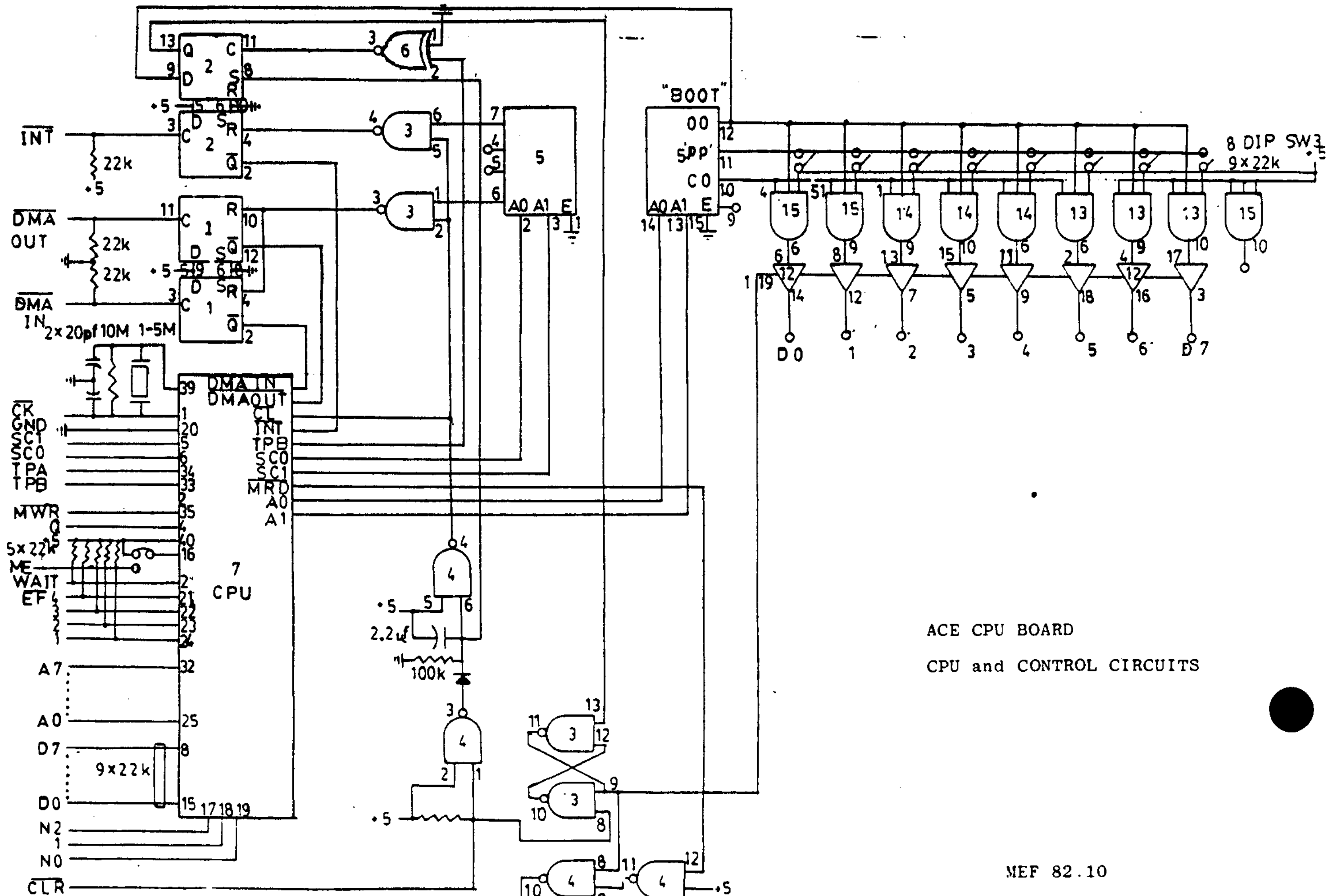


ACE CPU BOARD

PARTS PLACEMENT



ACE CPU BOARD
MEMORY DECODING
MEF 82.10



ACE CPU BOARD

CPU and CONTROL CIRCUITS

MEF 82.10

ACE CPU BOARD PARTS LIST

CPU Control & Boot

IC #	
1	4013
2	4013
3	4011
4	4093
5	4556
6	4077
7	1802/4/5/6
12	74C244
13	4073
14	4073
15	4073

Resistors

2 - 9 x 22K SIP or 18-22K 1/4 watt

10 - 22K 1/4 watt 5%

1 - 100K 1/4 watt

1 - 10 MEG 1/4 watt

Capacitors

1 - 2.2 mf tantalum

2 - 20 p.f. ceramic

3 - 10 mf tantalum (buss filters)

6 - 0.001 mf ceramic (bypass caps.)

Diode

1 - IN914

Crystal

1 - 1.0 meg to 5.0 meg

Switch

1 - 8 position dip

Memory

IC #	
8	4042
9	4042
10	4556
11	4556
17	EPROM/RAM
18	"
19	"
20	"

Resistors

2 22K 1/4 watt 5%

Switch

2 8 position dip

Port, UART, RS232C

IC #	
16	1853
21	1854
22	SMC-COM 8116 (P)
23	1852
24	1852
25	1488
26	1489

Resistors

4 22K 1/4 watt 5%

1 9 x 22K SIP or 9-22K 1/4 watt

Diodes

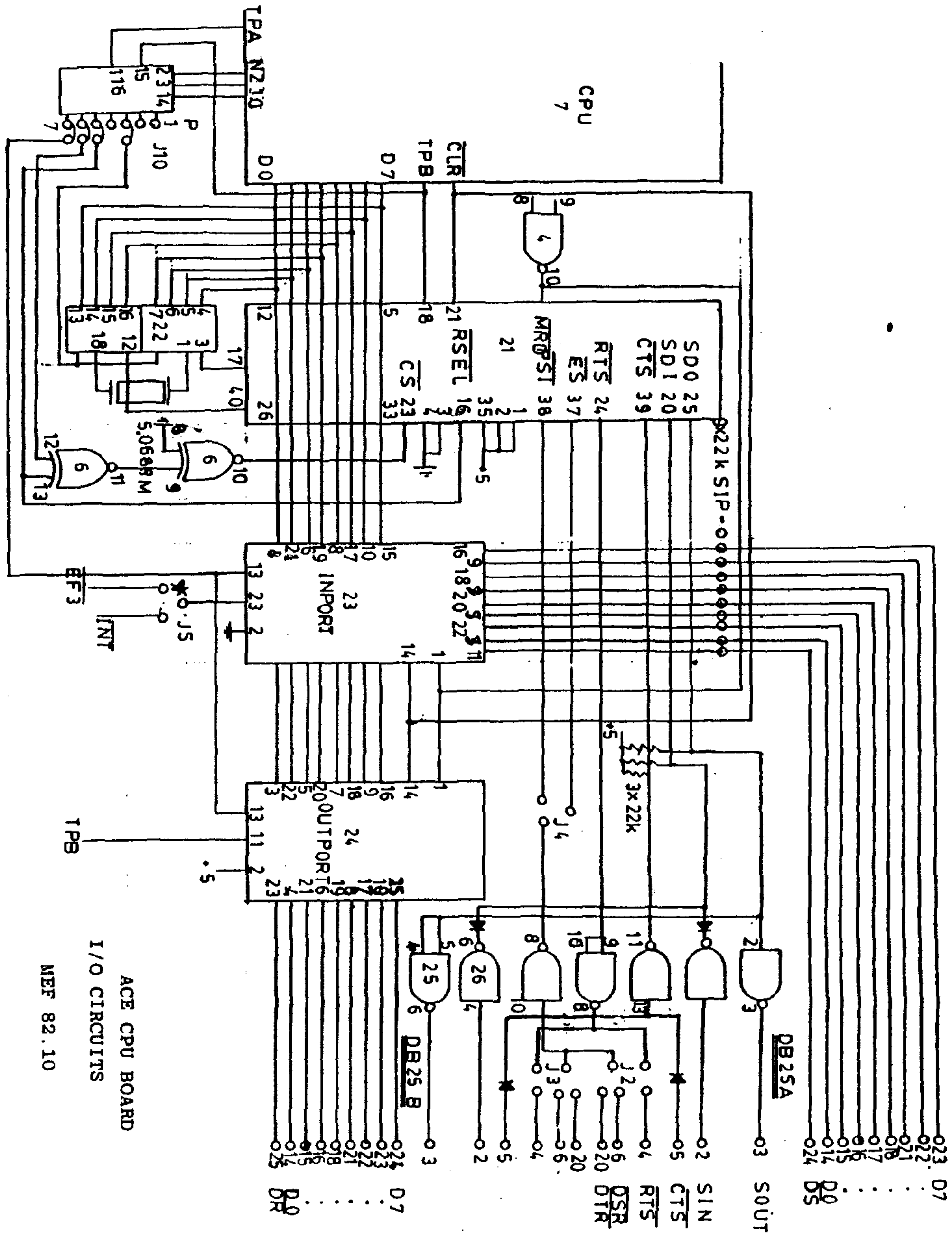
7 - IN 914

Crystal

1 5.0688 meg.

Connectors

2 Db25 Female
(wire solder type)



ACE CPU BOARD
I/O CIRCUITS

MEF 82.10

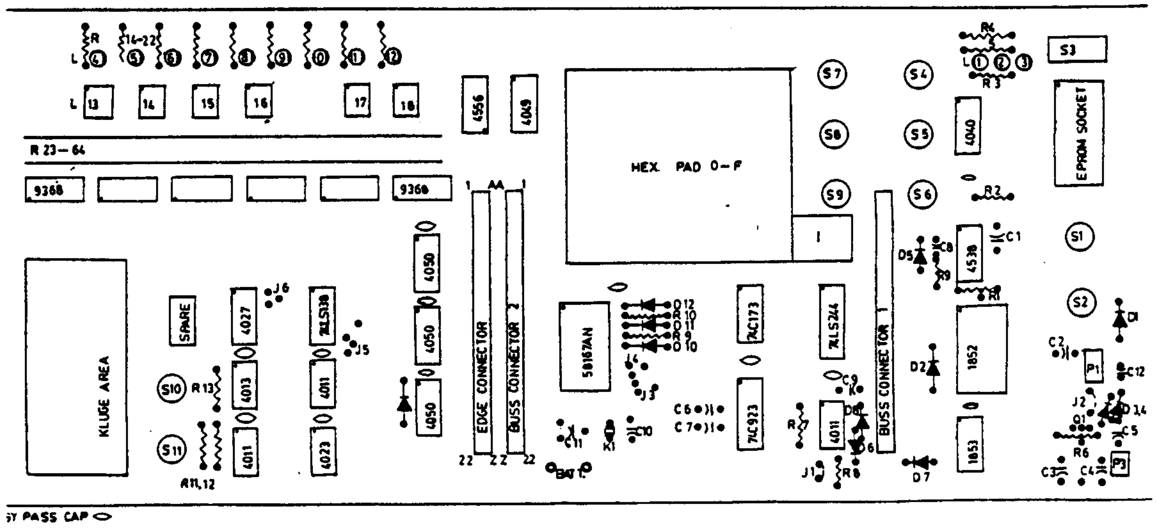
ACE FRONT PANEL

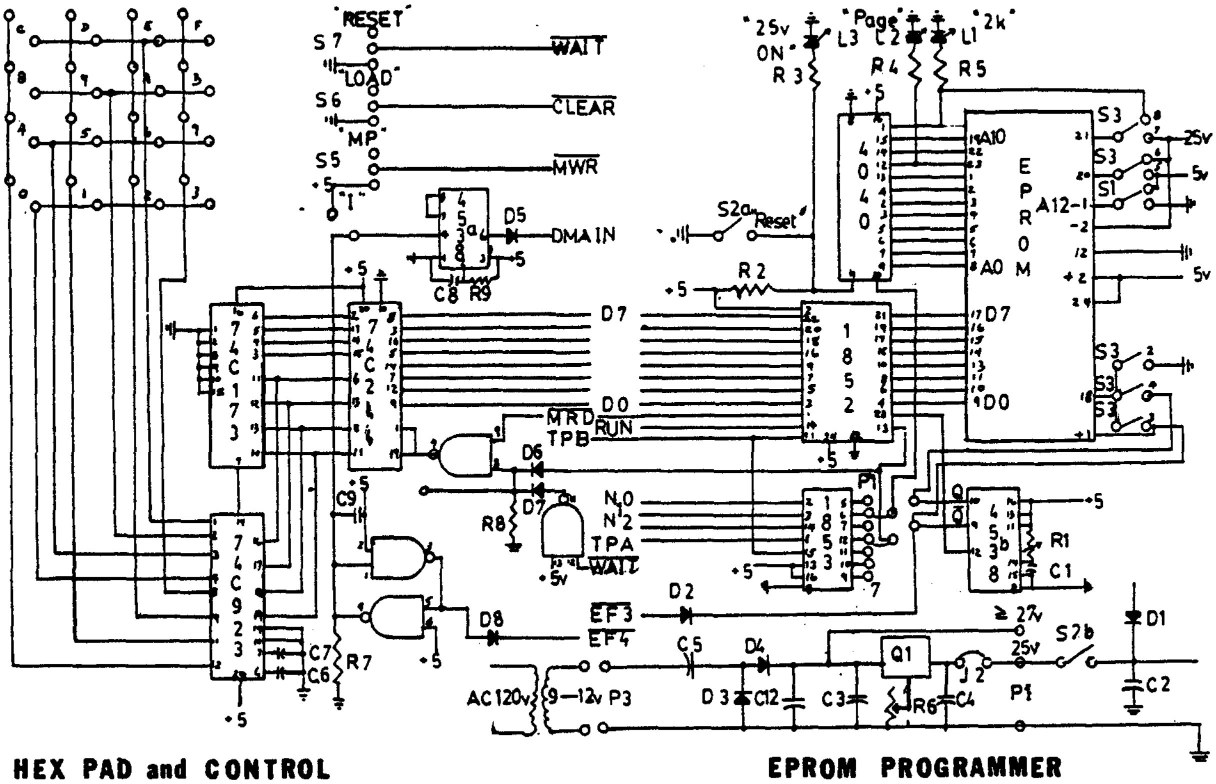
Size: 6" x 13.5"

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.

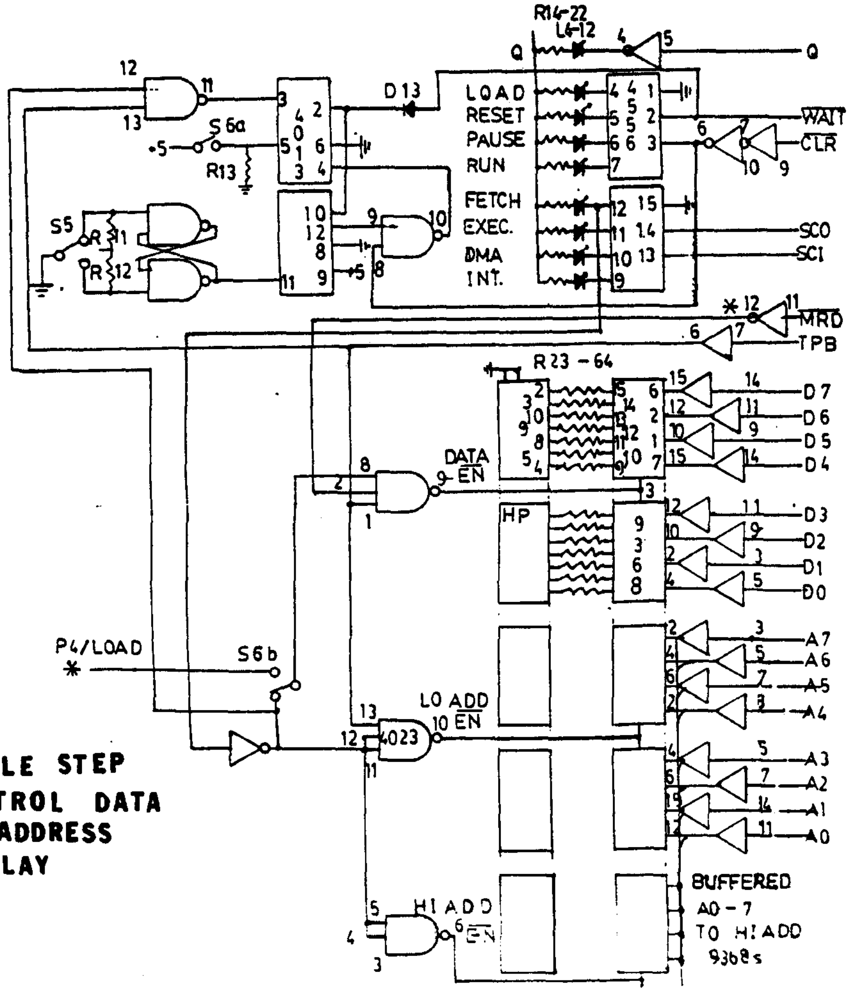




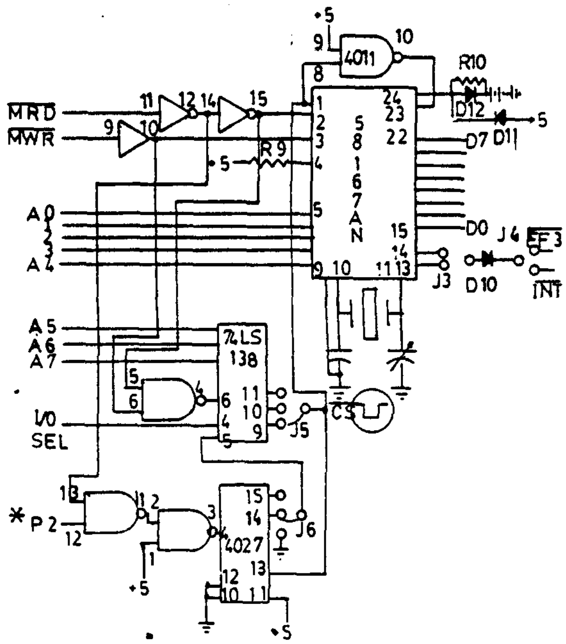
HEX PAD and CONTROL

EPROM PROGRAMMER

ACE FRONT PANEL



SINGLE STEP CONTROL DATA and ADDRESS DISPLAY



REAL TIME CLOCK II

ACE BACKPLANE AND I/O BOARD.

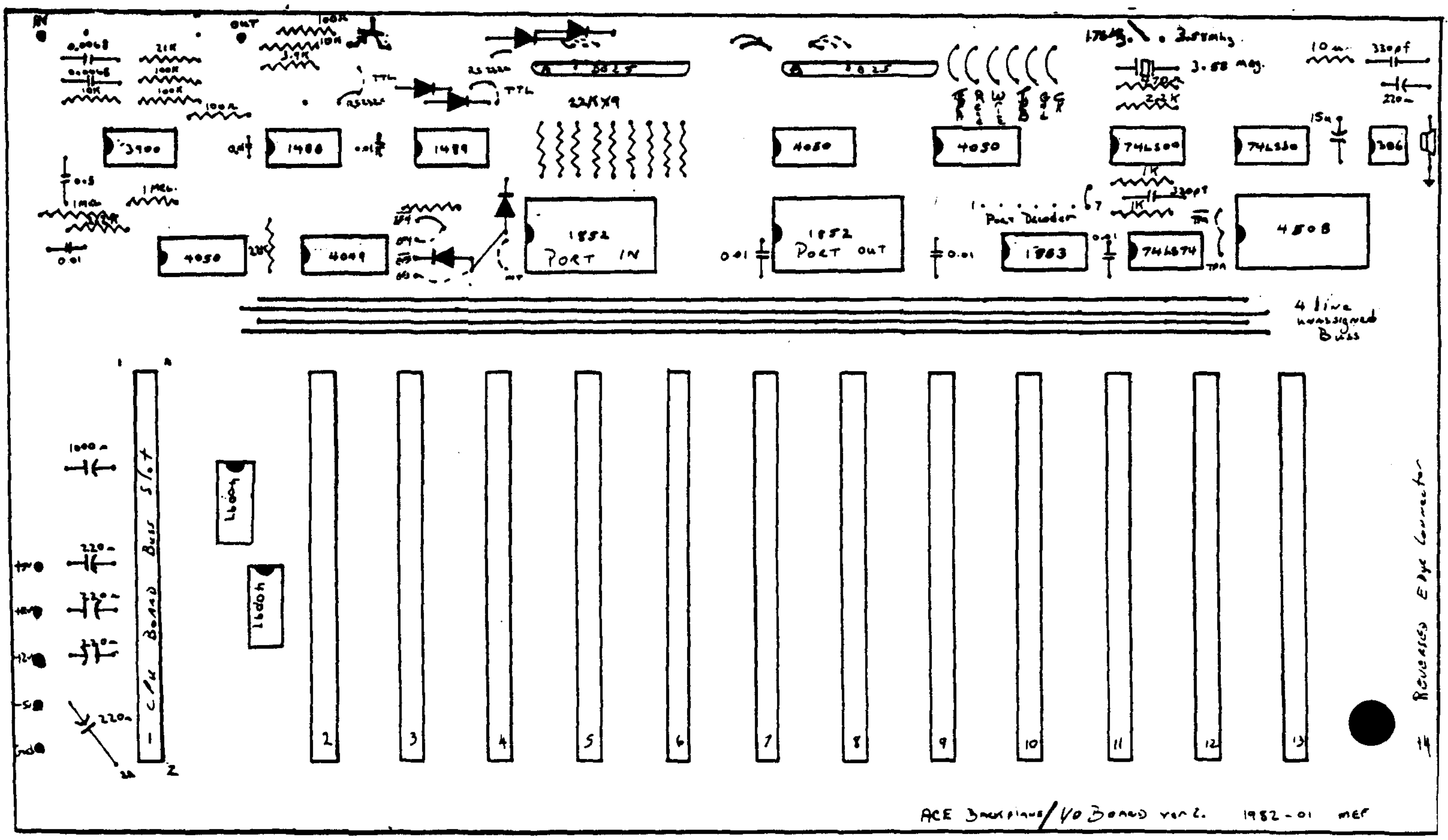
Sixe: 7.0" x 13.5"

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.

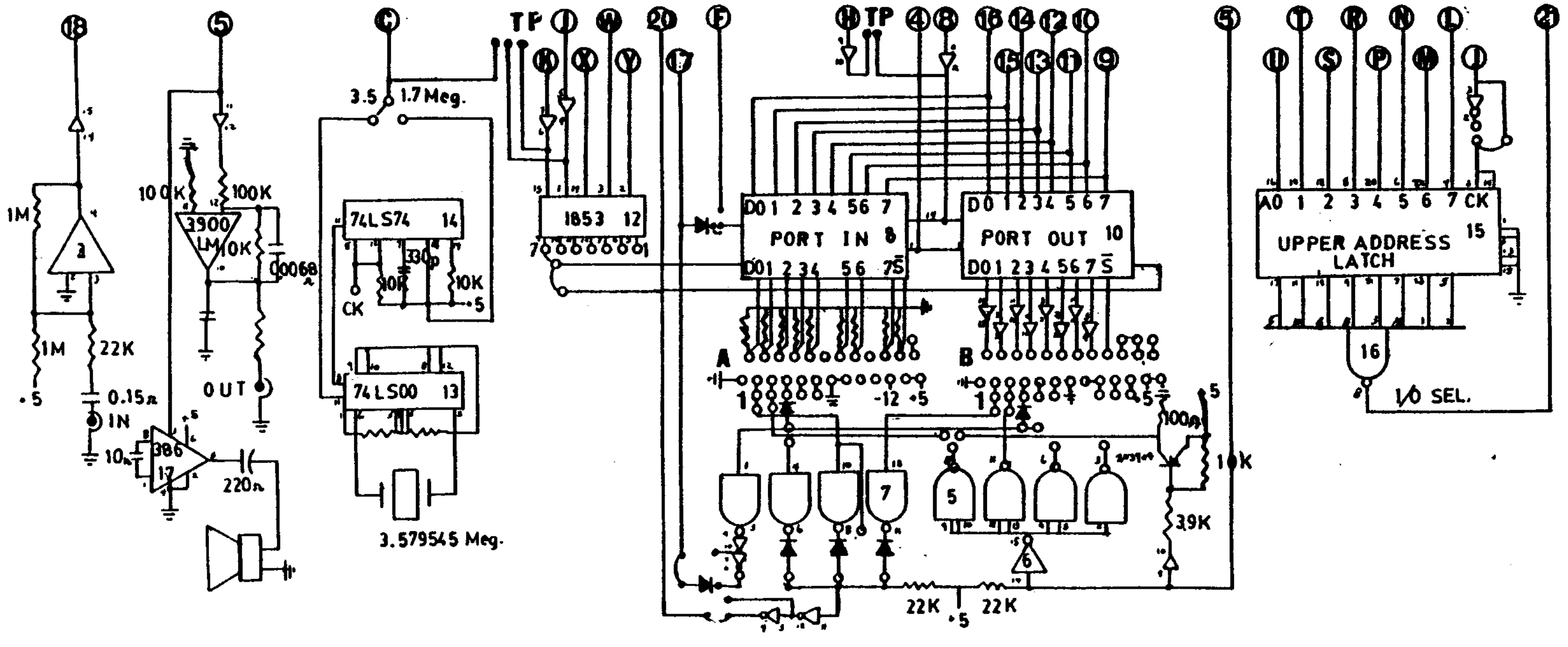
Power: -5v, -12v., Gnd.

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".



ACE Backplane/I/O Board ver 2. 1982-01 mef



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.

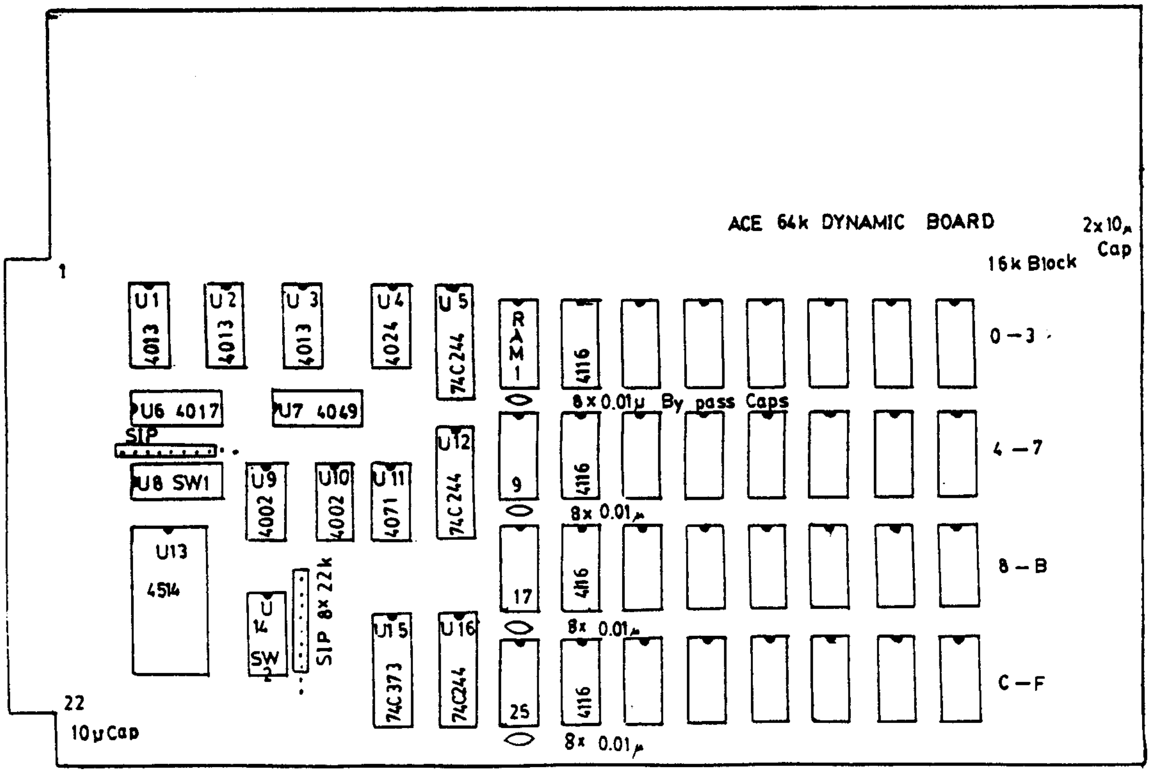
Size: 6.0" x 9.5"

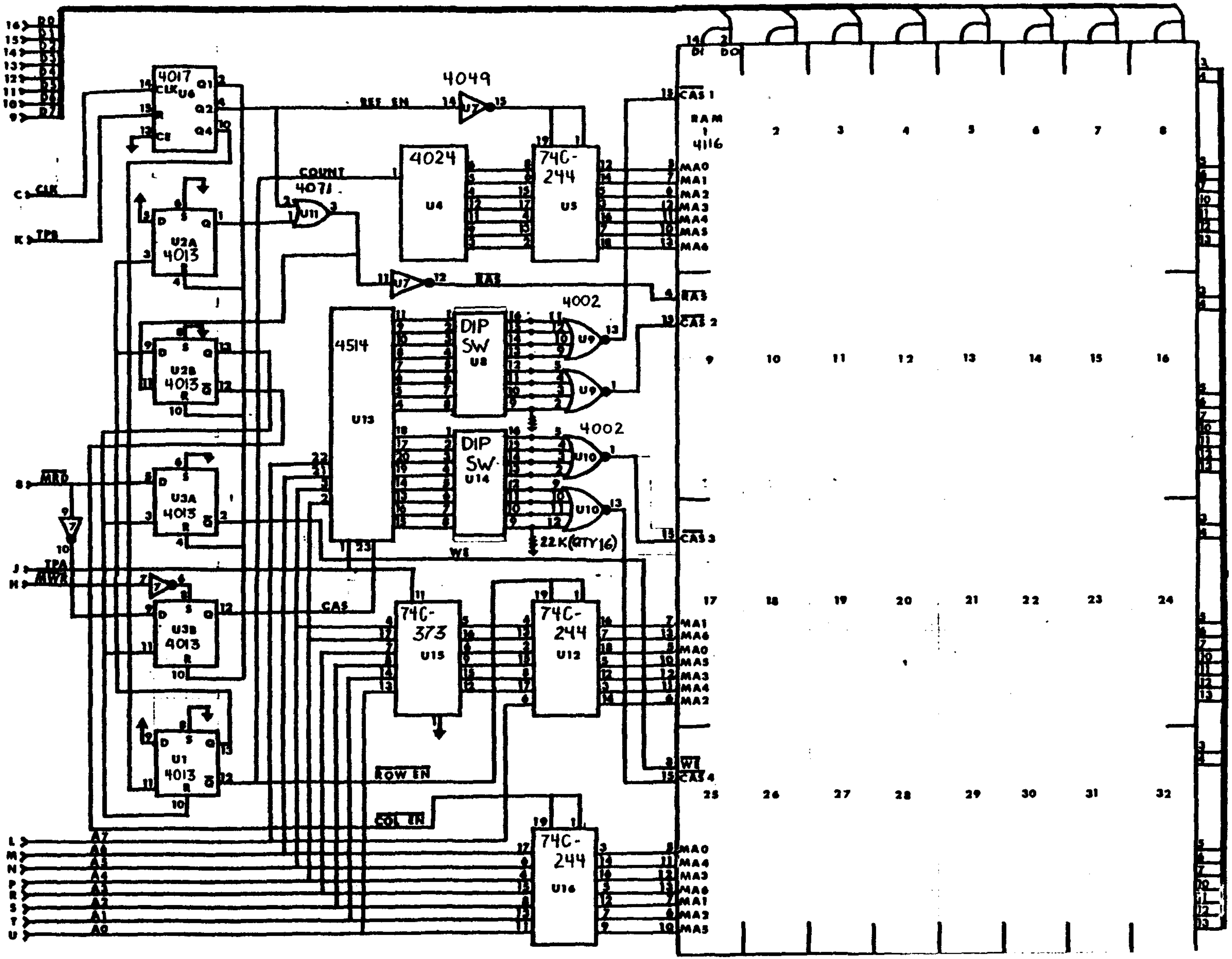
Function: to provide up to 64K of user RAM on the ACE configured buss. On board refresh independant of micro clock. RAM may be disabled in 4k blocks by sue of switches (S 1 and 2). May be populated in units of 16k. Flexible jumper provision at edge connector allows reconfiguration to other 44 pin configurations, 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
CLK
TPB
MRD
TPA
MWR
4017
U6
CLK
CE
Q1
Q2
Q4
4049
U7
REF EN
COUNT
4071
U11
4024
U4
74C-244
U5
U2A 4013
U2B 4013
U3A 4013
U3B 4013
U1 4013
U7
RAS
4514
U13
DIP SW
U8
DIP SW
U14
4002
U9
4002
U10
22K (QTY 16)
WE
CAS
74C-373
U15
74C-244
U12
ROW EN
COL EN
74C-244
U16
A7
A6
A5
A4
A3
A2
A1
A0
DI
DO
CAS 1
RAM 1 4116
MA0
MA1
MA2
MA3
MA4
MA5
MA6
RAS
CAS 2
CAS 3
WE
CAS 4
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32

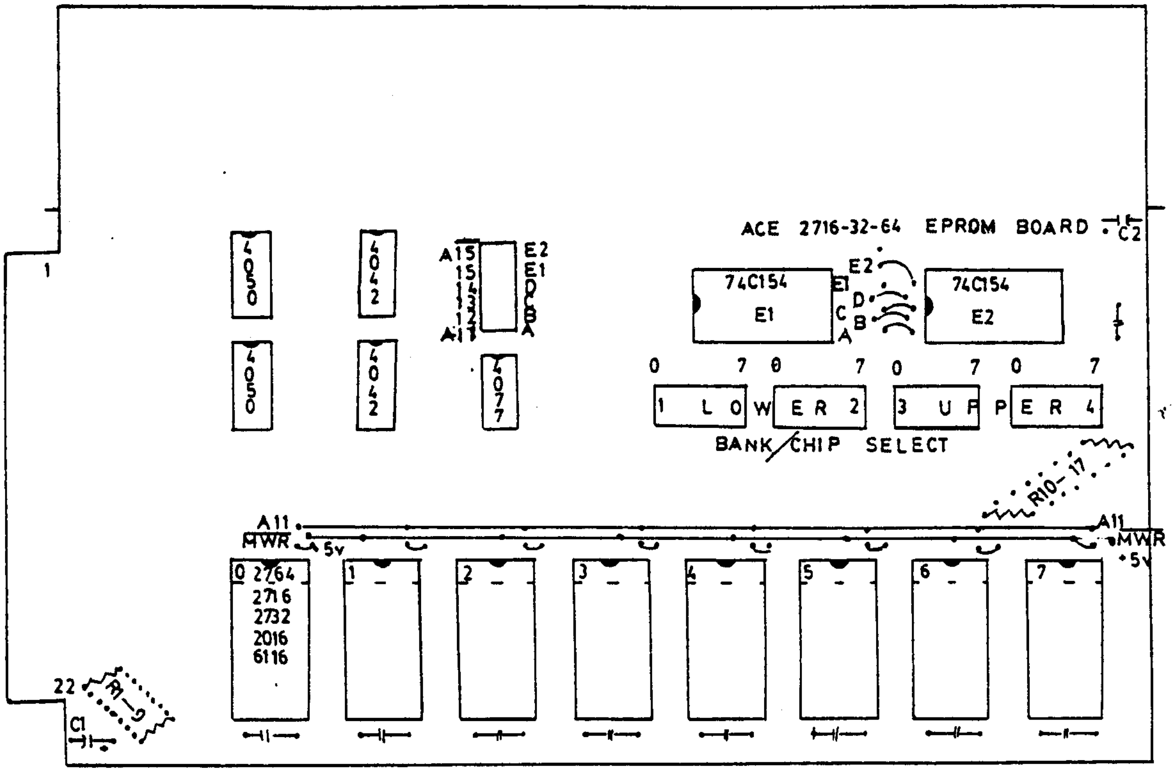
ACE 2716/32/64k EPROM BOARD.

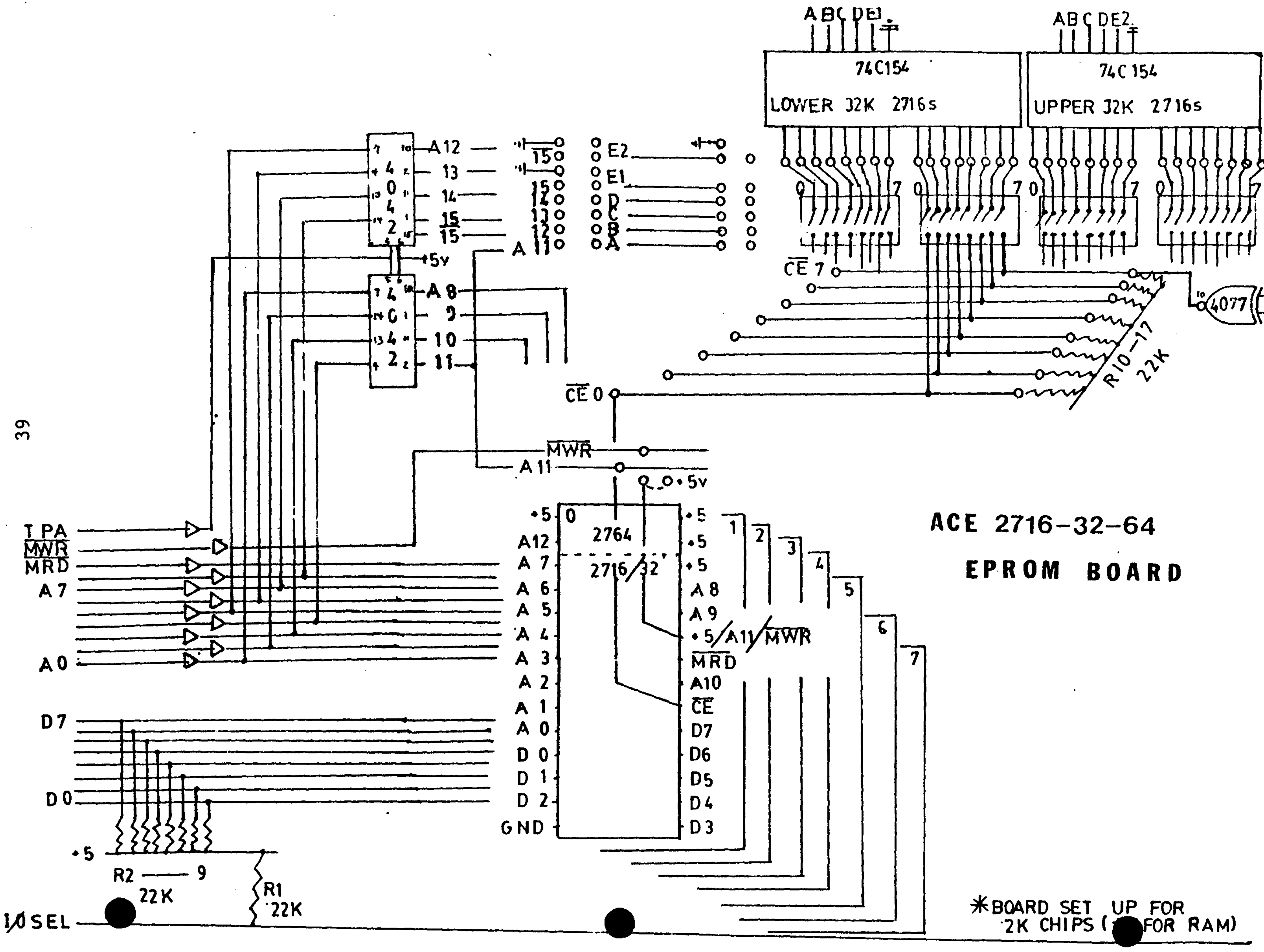
Size: 6.0" x 9.5"

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

BOARD SET UP FOR 2K CHIPS (FOR RAM)

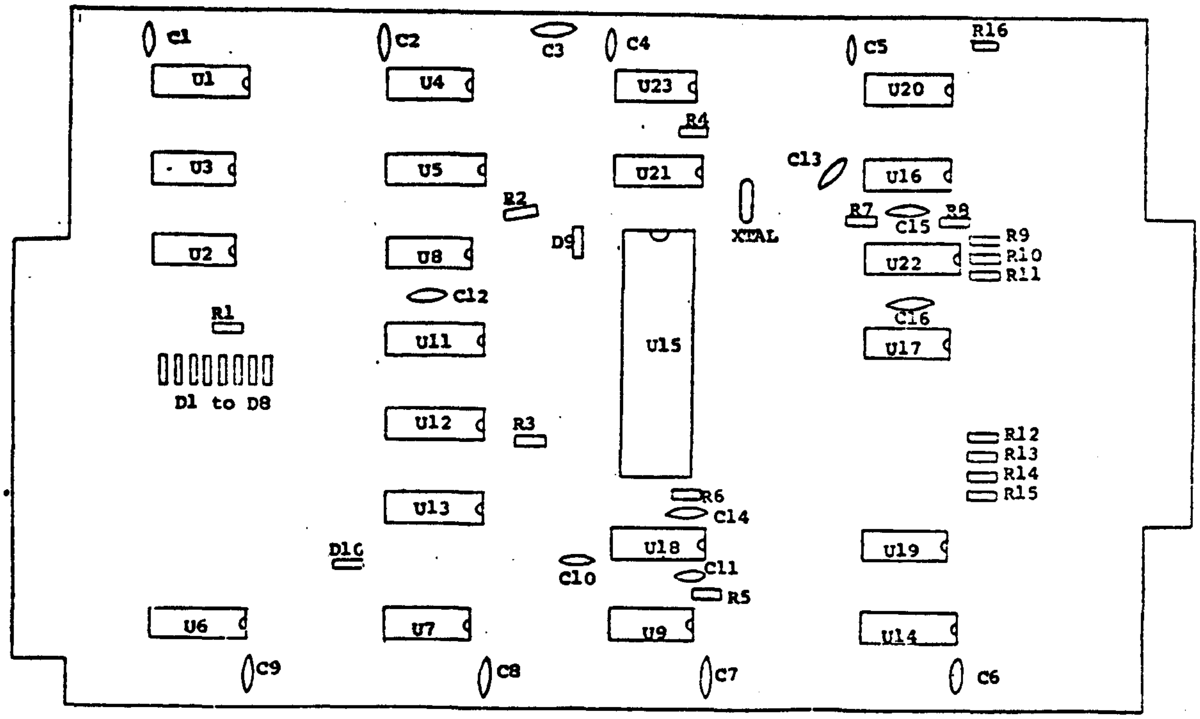
ACE DISK CONTROLLER BOARD

Size - 6.0" x 10.0"

Function - DMA oriented 8" Disk controller for the 1802.
Singe sided, single density WD 1771 Controller chip.
Designed to support two 8" Disks, jumperable Disk
Interface will accommodate any 8" Disk. Probably
could be modified to support 5.25" Disks.
ELF 11 users require DMA Adapter board for buss interface.

Documentation - assembly instructions, mini DOS, DOS exerciser program.

Power - ± 5 v., + 12v., Gnd.



CLUB COMMUNIQUE

Rev. 83.07.01

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	
VDU Board, Ver. 2		40.00	
64K Dynamic (4116) Board		50.00	
Netronics - Ace Adapter Board		25.00	
I/O Adapter for Backplane, Ver. 1		20.00	

Software

Fig FORTH - Netronics Cassette format (6K) 0000H		$10.00	
Tiny Pilot - Netronics Cassette format (2K) 0000H		$10.00	
SYMON - Netronics Cassette format (2K) C000H		$10.00	

Back Issues

"Defacto" Year 1 - 3 (Edited)		$20.00	
Year 4 Reprint		10.00	
Year 5 Reprint		10.00	
Year 6 Reprint		10.00	

Membership - Year 7

Current Year - Sept. '83 - Aug. '84
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
