Ipso Facto

ISSUE 27 JANUARY 1982

INDEX PAGE

PUBLICATION OF THE ASSOCIATION OF 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:	Mike Franklin	416-878-0740	Program Convener:	Bernie Murphy Bert Dekat	
Editors:	Fred Feaver Steve Carter Bob Siddall Tony Hill		Tutorial/Seminars:	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.

EDITORS CORNER

I am sure you are pleasantly surprised to see a newsletter so soon after receiving Number 26. There is a reason for our haste. The Canada Post Corporation has raised postage by over 100% effective January 1, 1982. This cost increase was beyond the club's budgeted increase, and quite frankly, beyond our ability to absorb. Since we operate on a annual subscription basis, we have no mechanism to generate more revenue from our members, so we must cut costs. By mailing this issue in 1981, we saved considerable postage costs. Unfortunately, it will not be enough. After much deliberation, the executive decided to restrict the size of newsletters to 42 pages (the size of issues 26 and 27). Forty-two pages will still permit the editors to create a broad ranging newsletter, which is just under the weight ceiling of level 2 first class mail rates. So far now, the 60 page encyclopedia newsletter is gone. We will still meet our commitment for 6 newsleters for this club year.

Your comments would be appreciated.

Club Products

The club is currently field testing a prototype EPROM board designed to accommodate 2716/32/64 single 5 volt supply EPROM (28 pin JEDEC standard). We expect to be able to offer the board for sale by 31 March 1982. Current projects underway include a redesigned club 44 pin buss back plane and a new micro processor board for the club buss.

Forth

Tony Hill made an excellent presentation on FORTH at the 8 December 1981 club meeting. His work on FORTH is well advanced and I should be able to report ordering information in the next newsletter.

Interested FORTH uses should be approaching FIG and Mountainview Press for appropriate documentation.

Best Article Issue 26: P. Liescheski - The Shroedinger Equation.

MEMBERS CORNER

FOR SALE.

- by Chuck Reid, 423 Huxley Ave., Sarnia, Ontario, N7S 4Z1

		Canadian Funds
2	- Netronics 4K Static Ram Boards	@ $50.00 each
1	- Netronics 4K Static Ram Board - memory bug, needs trouble shooting	@ $25.00
1	- Netronics Giant Board	@ $20.00
1	- Netronics AP-1 5 Amp Power Supply	@ $20.00
1	- SSM VB1C Video Display Board	@ $100.00
1	- Netronics VDB Video Display Board	@ $150.00

All boards are fully functional (except where noted) and are fully socketed (except Giant Board).

FOR SALE:

BASIC NETRONICS ELF II, factory assembled and tested, brand new condition, Netronics metal cabinet and cover 1/4 K Ram, UHF channel 33 modulator, 5V power supply, Quest interface board with hexdisplays up to 65K and all mode indicators factory assembled and tested, all original Netronics and RCA documentation, almost all Questdata and IPSO FACTO newsletter. - $200.00 or best offer.

Alain Jacynas, 3093 Allard, Montreal, P.Q., Canada, H4E 2M8 - Phone: DAY - 514-282-6530, EVENING - 514-761-7447

FOR SALE:

1	Netronics ELF II	$80.00
2	Netronics 4K Static Memory - each	$78.00
1	Case Computer	$25.00
1	5 Amp Power Supply	$30.00
1	GIANT BOARD I/O	$35.00
1	Netronics Protobard	$15.00
1	Netronics Video display board with ASCII K/B and RF Modulator only	$150.00

Most chips socketed professionally assembled. All manuals included. OR

Everything for $500.00 with Tiny Basic on Tape. Will consider trade for 8" disck drives.

Also available - Olivetti daisy wheel KSR terminal. RS-232 Interface inc. - $2,350.00 NEW

S. Carter, 8086 Islington Ave., Woodbridge, Ontario, L4L 1W3, Phone: 851-2921

FOR SALE:
- by Joe Matherly, Room 222A, 460 NE 215th St., Miami, Florida, 33179
  Tel: 305-653-4900

NETRONICS ELF II 4K Boards, fully socketed, all chips included, DIP Switch addressing, excellent working condition. All 3 for $130.00 or $50.00 each if sold separately. I'll pay postage.

FOR SALE:

Quest Super Elf with Expansion Board. All on-board options. Godbout 8K S100 video. Quest Super Color 6847-based board (partially assembled). ASC11 Keyboard. Full manuals and documentation, with some programs on cassette tapes. Quest Tiny Basic included.

All the above for $650.00 US. I will ship. My cost was over $1,000.

Also - ASR33 Teletype W/Modern. Everything works except tape reader unrealiable. $250 US. Included full documentation including schematics with diagrams. Also some spare parts.

A. D. Barksdale Garbee II, 1601 Clayton Avenue, Lynchburg, Virginia, 24503, USA, Phone: (804) 384-2470

NETRONICS FULL BASIC AND THE INFAMOUS $\overline{\text{EF 2}}$ LINE

- by J. Vaal, 6535 Velmar Dr., Ft. Wayne, Indiana, USA, 46811

After reading the last several issues of Ipso-Facto, I almost regretted that I ordered Netronics Full BASIC last Winter. Well, Full BASIC LEV III A1 arrived a couple weeks ago, complete with a user manual, chock full of errors (including the ending address of the program). My two biggest complaints about this system are: 1, The RPN format is not identical to HP calculators and 2, The modifications to the GIANT BOARD significantly degrade the performance of the cassette read hardware. Problem 1 can be evidenced by solving the equation:

$$X=5^2+6^2 .$$

Quickly you can see the difference in stack operations. Unfortunately I do not have a solution to this problem.

Problem 2 , which, I believe is actually two sub-problems, can be neatly solved. The first, as described by Mr. M.E. Franklin in Issue #17, where the math chip holds the $\overline{\text{EF2}}$ line low, occurs only when an error is detected by the math chip and the program is terminated or when the user terminates the program during math function execution. The capacitor / diode modification to the GIANT BOARD is not actually intended to and will not solve this problem. I believe, however, that the degradation of the cassette read hardware as a result of this modification is a far more significant problem. Figure 1 is a block diagram of the affected portions of the system when Full BASIC is installed. Capacitor Ca is added to the GIANT BOARD because the output of A12 (Pin 4) is normally low. Diode Db is a clamp on the output signal during cassette read and maintains the proper dc operating level. The problem with this modification, is that it partially defeats the purpose of the cassette read circuit (A12). This amplifier is intended to "square-up" the cassette signal, but capacitor Ca reduces the effectiveness of this circuit. Diodes Da (GIANT BOARD), D1, and D3 (BASIC board) comprise a "wired OR" so that the $\overline{\text{EF2}}$ line may be shared.

AUTO-SWITCH CIRCUIT OPERATION

Figure 2 is a block diagram of the system where the Full BASIC GIANT BOARD modifications are removed and the Auto-Switch modifications added to the Full BASIC board. This circuit (Figure 3) is essentially an automatic switch that connects either the cassette read or the Full BASIC $\overline{\text{EF2}}$ signals to the buss. The basic circuit consists of four components; U1 CD 4066 (quad bi-lateral switch), R1, R2, and C1. R3 and the LED are optional.

AUTO-SWITCH CIRCUIT OPERATION continued

SW1 acts as a buffer for the A12 GIANT BOARD $\overline{EF2}$ signal. When a cassette signal is present, C1 is charged to 5 volts which enables SW3 to put the cassette signal on the buss. SW2 is simply an inverter of the output of SW1 and turns off the Full BASIC $\overline{EF2}$ signals. (SW4). The optional LED turns on when the cassette line takes control. When no cassette signal is present, SW4 is turned on, and the Full BASIC board has control of the EF2 line.

This modification eliminates the ac coupling of the cassette signal to the buss and allows a cassette read even when the Full BASIC board would be holding EF2 low. It should be noted, however, that after the program is loaded, it still may be necessary to enter "PR CL#" to resume normal operation of Full BASIC. This is because the "LOAD" routine does not reset the math chip.

This circuit does give priority to cassette operations and therefore the cassette recorder should not be operated during Full BASIC program execution.

CONSTRUCTION DETAILS

The circuit shown in **Figure** 3 may be readily added to the Full BASIC P.C. board in the area reserved for user hardware in the lower right hand corner of the board. The following modifications are necessary.

1) Remove the .1uf capacitor and the two diodes that were added to the GIANT BOARD for Full BASIC.

2) Connect a wire from A12 Pin 4 (where the capacitor was) to Pin 84.

3) Connect a wire on the mother board between pins 84 of the GIANT and Full BASIC board sockets.

4) On the Full BASIC board, cut the trace to Pin 70 after the junction of D1 and D3.

5) Wire up the Auto-Switch circuit as shown in Figure 3.

6) The total system should now be wired as shown in Figure 2.

At this point, one note of caution is in order. If the Full BASIC board is removed from the system, Pin 70 on the GIANT BOARD must be reconnected for cassette read operation. I have been using Full BASIC with this modification for several days now and have encountered no problems. As long as one remembers that the cassette will take priority over the math chip (as described earlier) there should be no problems. One interesting side effect of this modification is that the LED acts as a cassette signal present indicator, which is cute if not functional.

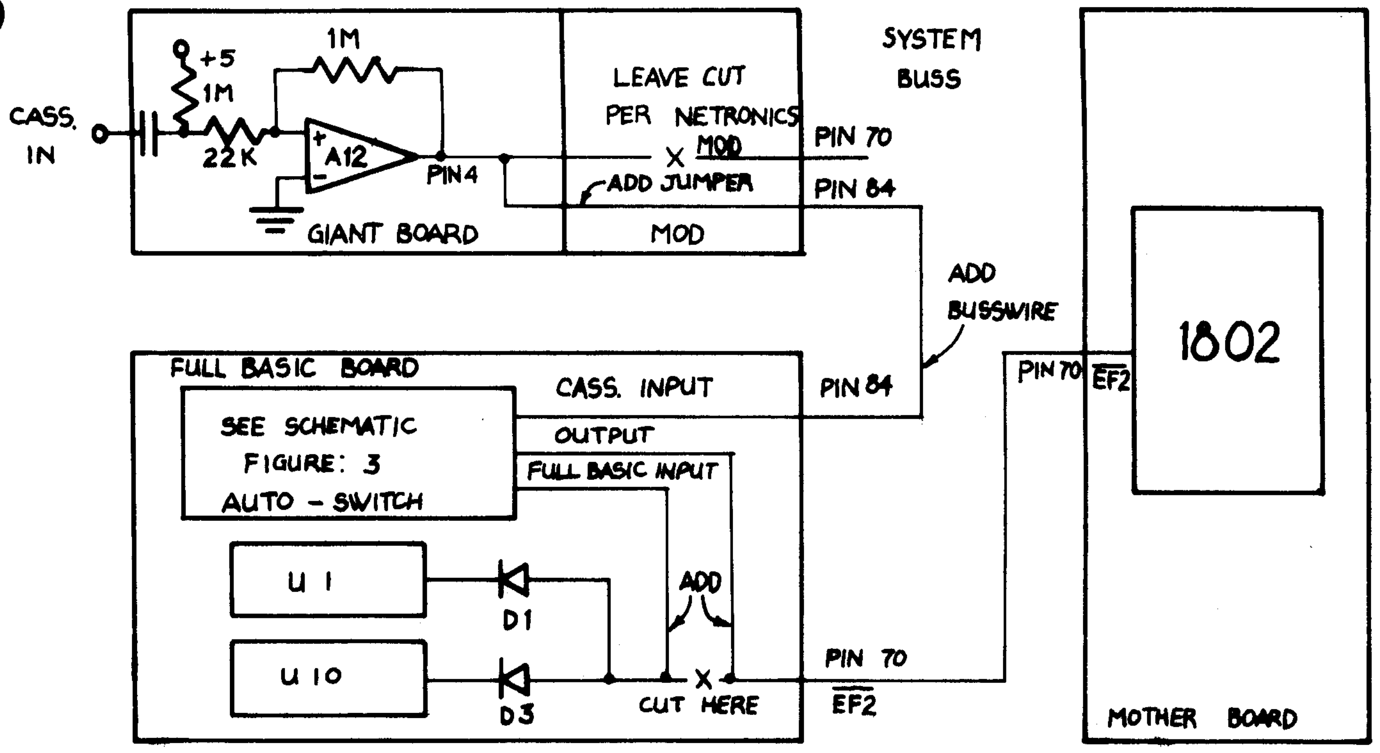


FIGURE 2: SYSTEM WITH AUTO - SWITCH MOD

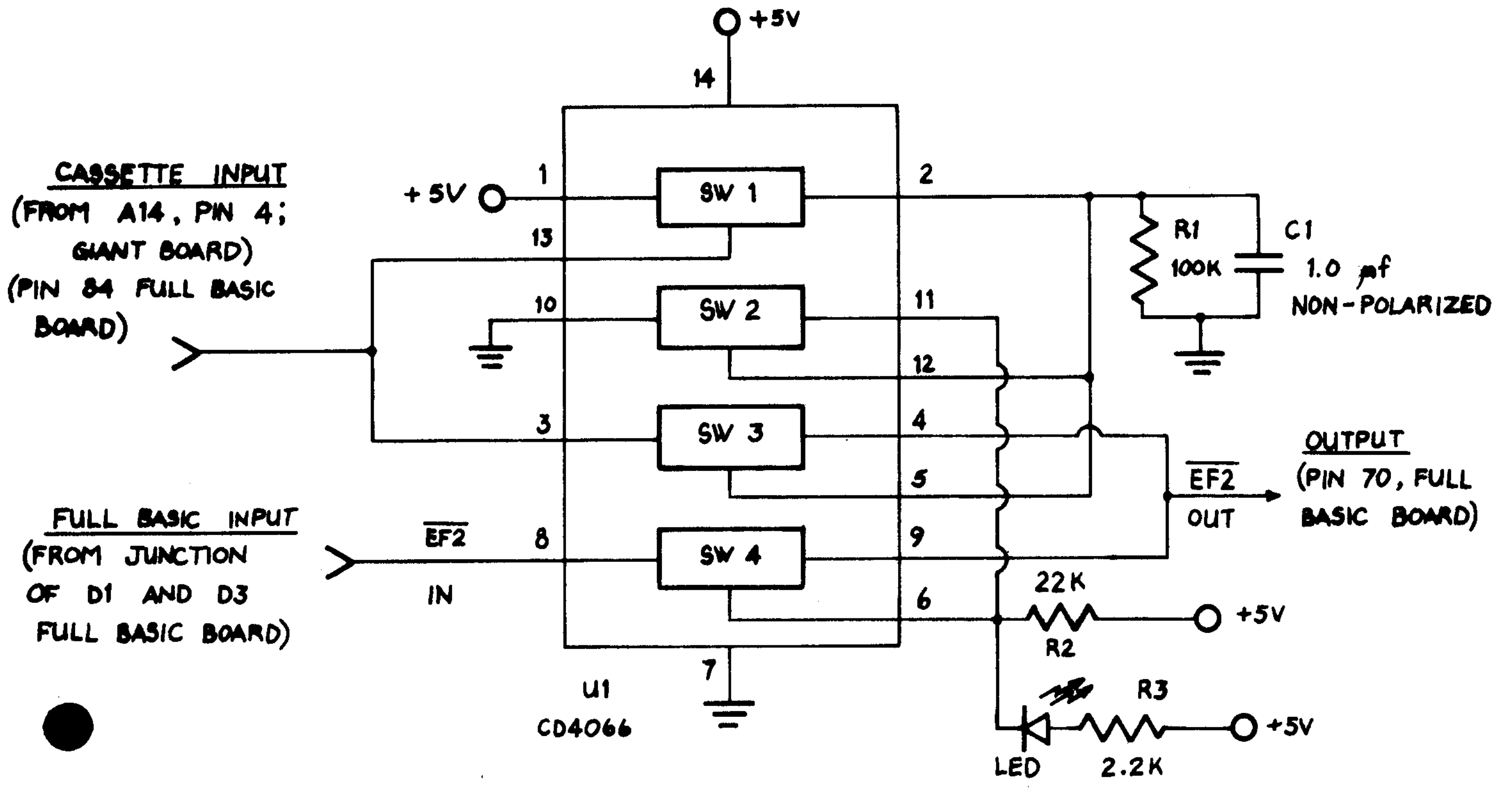


FIGURE 3: $\overline{\text{EF2}}$ LINE AUTO-SWITCH

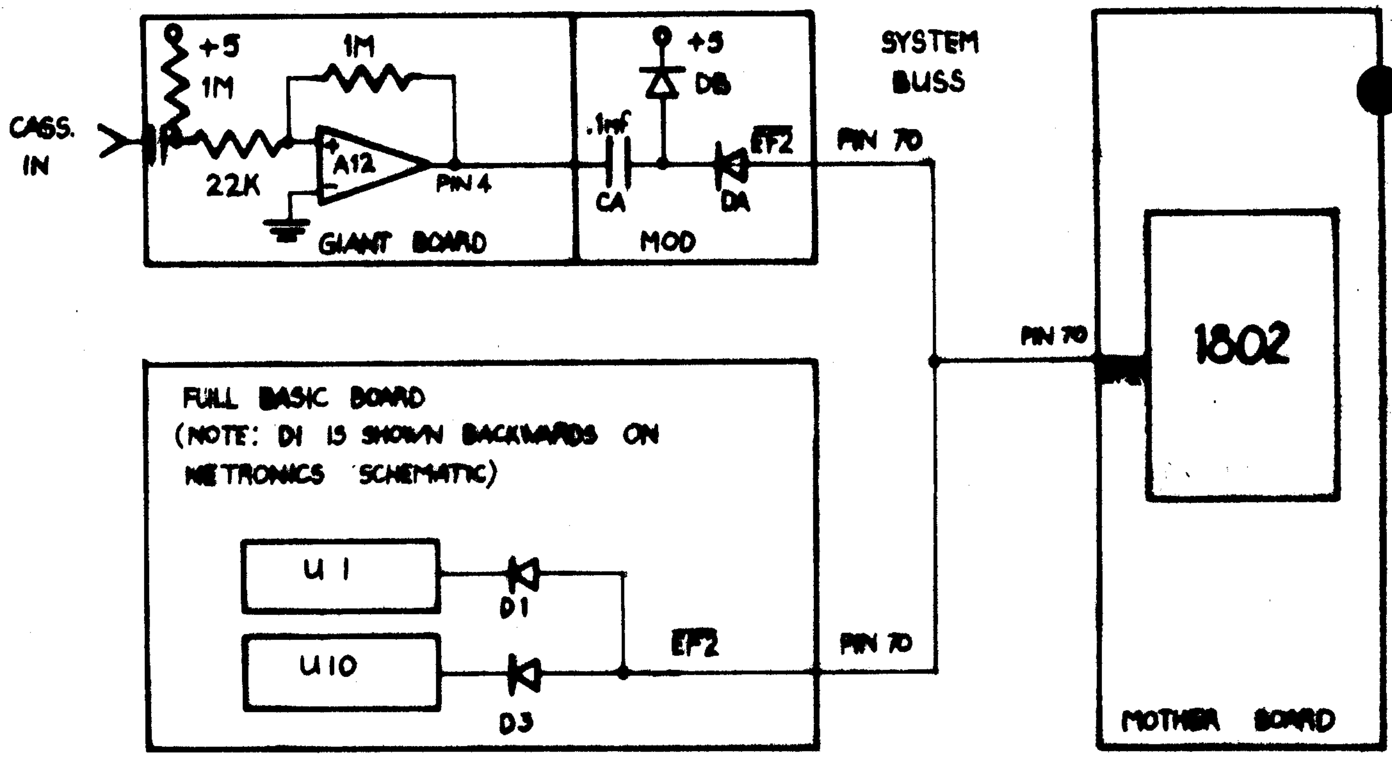


FIGURE 1: SYSTEM WITH NETRONICS MOD

TINY BASIC PROGRAMS

-by G. Caughman - 3795 Somerset Dr., Marqetta, Georgia, USA, 30064

DECIMAL TO HEX CONVERSION ROUTINE

```
10 REM INPUT DECIMAL VALUE AND THIS ROUTINE WILL
20 REM CONVERT IT TO AN EQUIVALENT HEX VALUE
100 INPUT X
110 LET I=X/16
120 U=X-16*I
130 LET J=I/16
140 T=I-16*J
150 LET K=J/16
160 S=J-16*K
170 LET L=K/16
180 R=K-16*L
400 LET V=R
410 GOSUB 500
420 LET V=S
430 GOSUB 500
440 LET V=T
450 GOSUB 500
460 LET V=U
465 GOSUB 500
470 GOSUB 1000
```

```
 500 IF V<10 GOTO 570
 510 IF V=10 PRINT "A";
 530 IF V=11 PRINT "B";
 550 IF V=12 PRINT "C";
 552 IF V=13 PRINT "D";
 554 IF V=14 PRINT "E";
 556 IF V=15 PRINT "F";
 560 RETURN
 570 PRINT "V";
 580 RETURN
1000 PRINT
1010 END
```

TINY BASIC HEX TO DECIMAL ROUTINE

```
 10 LET A=10
 20 LET B=11
 30 LET C=12
 40 LET D=13
 50 LET E=14
 55 LET F=15
 60 PRINT "INPUT FOUR HEX DIGITS EACH FOLLOWED BY COMMAS";
 65 PRINT "EXCEPT LAST DIGIT.(MAX. 7,F,F,F)"
 70 INPUT U,T,V,W
 75 U=U*16*16*16
 80 T=T*16*16
 85 V=V*16
 90 X=U+T+V+W
 95 PRINT X
100 END
```

```
5     PRINT "MEMORY DISPLAY PROGRAM"
6     PRINT
7     PRINT "ENTER THE STARTING ADDRESS AS FOLLOWS:  N,N,N,N.";
11    PRINT "WHERE N EQUALS EACH HEX DIGIT.  (MAX  7,F,F,F)"
20    GO SUB  1010
25    LET Y= X
30    PRINT "ENTER LAST ADDRESS THE SAME WAY"
35    GO SUB  1010
40    LET Z= X
50    REM CLEAR SCREEN AND SPACE
51    PLOT  (12)
52    PRINT
56    LET X=Y
57    GO SUB  110
58    LET W=PEEK (Y)
59    Y= Y+1
65    LET X=W
72    REM SET BYTE FLAG FOR HEX OUTPUT ROUTINE
73    LET P=1
75    GO SUB 110
80    IF Y<Z+1  THEN GO TO 56
85    END
```

```
100  REM HEX OUTPUT ROUTINE
110  LET I=X/16
120  U=X-16 * I
130  LET J=I/16
140  T=I-16*J
150  LET K=J/16
160  S=J-16*K
170  LET L=K/16
180  R=K-16*L
400  LET V=R
402  IF P=1 GOTO 435
410  GOSUB 500
420  LET V=S
430  GOSUB 500
434  REM RESET BYTE FLAG IF SET
435  LET P=0
440  LET V=T
450  GOSUB 500
460  LET V=U
465  GOSUB 500
466  REM ALLOW TWO SPACES
470  PRINT "  ";
475  RETURN
500  IF V<10 GOTO 570
510  IF V=10 PR "A";
530  IF V=11 PR "B";
550  IF V=12 PR "C";
552  IF V=13 PR "D";
554  IF V=14 PR "E";
556  IF V=15 PR "F";
560  RETURN
570  PRINT V;
580  RETURN
1000 REM HEX TO DECIMAL ROUTINE
1010 LET A=10
1020 LET B=11
1021 LET C=12
1022 LET D=13
1023 LET E=14
1024 LET F=15
1030 INPUT U,T,V,W
1040 U=U*16*16*16
1050 T=T*16*16
1060 V=V*16
1070 X=U+T+V+W
1090 RETURN
```

THE MEGABYTE ELF

- by R. Siddall - 40 Cadillac Ave., Downsview, Ontario, Canada, M3H 1S2

INTRODUCTION

UP TO NOW, FEW HOBBYISTS HAVE FOUND THE 64K ADDRESSIBILITY LIMIT OF THE 1802 TO BE MUCH OF A PROBLEM. THE SITUATION MAY SOON CHANGE, HOWEVER. WITH THE APPEARENCE ON THE MARKET OF 64K X 1 RAM CHIPS, AND THE CONSTANTLY DECREASING COST OF OTHER LESS CAPACIOUS MEMORY CHIPS, IT MAY SOON BE NECESSARY TO FIND A WAY TO EXPAND THE USABLE MEMORY SPACE. TO ADDRESS THE PROBLEM (IF YOU WILL PARDON THE PUN) I HAVE DEVISED A MEMORY MANAGEMENT SCHEME THAT WOULD PERMIT AN 1802-BASED COMPUTER TO USE UP TO 1 MEGABYTE OF MEMORY, IN 64K SEGMENTS. OTHER BENEFITS OF MY SCHEME WOULD BE A MEMORY PROTECTION CAPABILITY, AND AN ABILITY TO SEPARATE THE PROGRAM SPACE FROM THE ADDRESS SPACE.

THIS SYSTEM IS AT PRESENT MERELY 'THEORETICAL', BUT IT IS RELATIVELY STRAIGHTFORWARD, AND I HOPE TO MAKE A BREADBOARD IMPLEMENTATION SOMEDAY. I ALSO INVITE OTHERS TO TRY IT OUT AND LET ME KNOW WHAT PROBLEMS I HAVE FAILED TO FORSEE.

ACCESSING 1 MEGABYTE REQUIRES A 20 BIT ADDRESS, 4 BITS MORE THAN IS PROVIDED BY THE 1802 ARCHITECTURE. IN MY SCHEME, TWO 4-BIT 'NYBBLES' WOULD BE LATCHED OFF THE DATA BUS BY AN I/O INSTRUCTION IN THE PROGRAM BEING EXECUTED. ONE OF THESE WOULD BE USED AS THE 'SEGMENT ADDRESS' (I.E. ADDRESS BITS 17-20) WHENEVER THE PROGRAM COUNTER REGISTER IS BEING USED TO ACCESS MEMORY: DURING THE FETCH CYCLE, AND WHENEVER AN 'IMMEDIATE' OR BRANCH INSTRUCTION IS BEING EXECUTED. THE OTHER SEGMENT ADDRESS IS USED FOR ALL OTHER MEMORY ACCESSES INCLUDING DMA REQUESTS. THE PROGRAMMER MAY CHOOSE TO MAKE THESE TWO SEGMENT ADDRESSES POINT TO THE SAME SEGMENT, OR HE CAN HAVE A 64K PROGRAM SPACE AND A 64K DATA SPACE SIMULTANEOUSLY ACCESSIBLE.

HARDWARE

FIGURE I SHOWS THE HARDWARE REQUIRED. IC1 IS A STANDARD 1852 I/O PORT CONTROLLED BY AN I/O LINE FROM THE CPU. IT HOLDS ONTO THE SEGMENT ADDRESSES SENT TO IT BY THE CPU WHEN THE APPROPRIATE OUTPUT INSTRUCTION IS EXECUTED. AN RC CIRCUIT ON THE CLEAR PIN COULD BE USED TO SET BOTH SEGMENT ADDRESSES TO 0 AT POWER ON.

IC2, IC3, AND IC4 CONSTITUTE A LOGICAL ARRAY WHICH DISTINGUISHES WHETHER AN INSTRUCTION BEING FETCHED ON THE DATA BUS WILL REQUIRE DATA TO BE READ FROM THE PROGRAM SPACE (I.E. VIA REGISTER P) OR THE DATA SPACE (VIA REGISTER X OR ANY REGISTER OTHER THAN P). THE TRUE/FALSE OUTPUT OF THIS LOGICAL ARRAY IS HELD AFTER A FETCH CYCLE IN ONE HALF OF A 4013 DUAL D LATCH (IC6). I CALL THIS CIRCUIT (FIG. 2) THE 'INSTRUCTION/DATA DISCRIMINATOR'.

IC5 IS USED TO DISCRIMINATE BETWEEN A FETCH AND AN EXECUTE CYCLE. THE OTHER HALF OF IC6 PUTS OUT A TRUE/FALSE SIGNAL (AND ITS INVERSE) DEPENDING

ON WHICH OF THE SEGMENT ADDRESSES IS TO BE USED IN THE SUBSEQUENT FETCH. IC7 IS A 4019 4 OF 8 SELECTOR WHICH PUTS OUT EITHER ONE OR THE OTHER SEGMENT ADDRESS FROM IC 1 DEPENDING ON THE OUTPUT OF IC6. IC8 IS A 4 TO 16 LINE DECODER WHICH WILL SELECT THE SEGMENT OF MEMORY REQUIRED.

I HAVE SPECIFIED CMOS ICS THROUGHOUT THE CIRCUIT, BUT I FULLY REALIZE THAT TIMING WILL PROBABLY BE A CRITICAL FACTOR, AND SOME OR ALL OF THE ICS MAY HAVE TO BE STTL OR EVEN ECL EQUIVALENTS TO THE CHIPS INDICATED TO MAKE THE SYSTEM WORK.

SOFTWARE

THE 1852 I/O PORT WHICH HOLDS THE SEGMENT ADDRESSES CAN BE TIED TO ANY AVAILABLE I/O LINE ON THE 1802. IN WHAT FOLLOWS I ASSUME IT IS TIED TO NO. AN 'OUT1' (61) INSTRUCTION CAN THEN BE USED TO WRITE THE SEGMENT ADDRESSES TO THIS PORT. THE HIGH ORDER 4 BITS OF THE BYTE STORED IN THE PORT WOULD CONTAIN THE PROGRAM SEGMENT WHILE THE LOW ORDER 4 BITS WOULD CONTAIN THE DATA SEGMENT.

THE DATA SEGMENT CAN EASILY BE CHANGED AT ANY TIME UNDER PROGRAM CONTROL. LISTING I GIVES A SMALL PROGRAM THAT WOULD COPY DATA FROM SEGMENT 1 TO SEGMENT 2.

CHANGING THE PROGRAM SEGMENT WOULD BE A LITTLE MORE DIFFICULT, SINCE THE PROGRAM COUNTER WOULD BE UNAFFECTED BY THE CHANGE OF SEGMENT ADDRESS. FOR EXAMPLE, IF THE 'OUT1' INSTRUCTION THAT CHANGED THE SEGMENT ADDRESS WERE AT LOCATION 001F0, AND THE SEGMENT WERE CHANGED FROM 0 TO 2, THE NEXT INSTRUCTION EXECUTED WOULD BE THE ONE AT LOCATION 201F1. TO GET AROUND THIS, A PROGRAM SEGMENT CHANGE CONVENTION OF SOME KIND WOULD HAVE TO BE SET UP. MY SUGGESTION IS THAT EVERY SEGMENT USED AS A PROGRAM SEGMENT HAVE A 'SEP R0' INSTRUCTION IN LOCATION 0000 AND AN 'OUT1' INSTRUCTION IN LOCATION FFFF (THIS COULD BE IMPLEMENTED IN HARDWARE). THEN, TO CHANGE PROGRAM SEGMENTS, THE PROCEDURE WOULD BE AS FOLLOWS:

- SWITCH THE PC TO A REGISTER OTHER THAN R0,

- POINT THE X-REGISTER TO A BYTE CONTAINING THE TWO NEW SEGMENT ADDRESSES,

- POINT REGISTER R0 TO THE ENTRY POINT OF THE PROGRAM TO BE EXECUTED IN THE NEW SEGMENT,

- BRANCH TO LOCATION FFFF.

CONTROL WOULD THEN JUMP TO THE 'OUT1' INSTRUCTION AT LOCATION FFFF, THE SEGMENT WOULD CHANGE. THE NEXT INSTRUCTION EXECUTED WOULD BE THE 'SEP R0' INSTRUCTION AT LOCATION 0000 OF THE NEW SEGMENT, CAUSING A BRANCH TO THE DESIRED PROGRAM.

TO TAKE ADVANTAGE OF THE SEPARATE ADDRESS SPACE, SPECIAL ASSEMBLER AND BASIC SOFTWARE WOULD HAVE TO BE DEVELOPED. FOR EXAMPLE, ADDRESS LABELS WOULD HAVE TO BE MARKED SOMEHOW AS TO WHETHER THEY WERE ADDRESSES IN THE PROGRAM OR DATA SPACE. EXISTING SOFTWARE WOULD STILL RUN, BUT IT WOULD BE CONFINED TO THE USUAL 64K.

MEMORY PROTECTION

INHERENT IN THIS SCHEME IS A USEFUL FORM OF MEMORY PROTECTION. IF THE PROGRAM AND DATA SEGMENTS ARE DIFFERENT, IT IS IMPOSSIBLE FOR THE PROGRAM TO BE CLOBBERED BY A STRAY DATA POINTER OR FOR THE SYSTEM TO ATTEMPT TO EXECUTE DATA. FURTHERMORE, DATA IN SEGMENTS OTHER THAN THE TWO ACTIVE ONES CANNOT BE ACCESSED AT ALL, UNLESS AN ERRONEOUS 'OUT 1' INSTRUCTION IS EXECUTED.

FRONT PANEL FUNCTIONS

WITH A BIT OF ADDITIONAL HARDWARE, A FEW USEFUL FRONT PANEL FEATURES COULD BE ADDED. FOR EXAMPLE

- -THE OUTPUT OF IC6 COULD BE USED TO LATCH OUT THE CURRENT TRUE PROGRAM COUNTER FROM THE ADDRESS LINES.
- -THE ACTIVE SEGMENT ADDRESS(ES) COULD BE DISPLAYED ON THE FRONT PANEL.
- -THE PROGRAM AND DATA SEGMENTS COULD BE CONTROLLED FROM THE FRONT PANEL BY OVERRIDING THE OUTPUT OF IC7.

OTHER CONSIDERATIONS

OF COURSE, IT WOULD NOT BE NECESSARY TO ACTUALLY HAVE A MEGABYTE OF STORAGE, OR EVEN 64K, TO TAKE ADVANTAGE OF THIS SCHEME. IN THE SYSTEM I ENVISAGE, SEGMENT 0 WOULD CONSIST ENTIRELY OF ROM, AND WOULD CONTAIN THE OPERATING SYSTEM (STARTUP PROGRAMS, MONITOR, INTERPRETERS, MATHEMATICAL SUBROUTINES, ETC.). SEGMENT 1 WOULD BE RAM, BUT PART OF IT WOULD BE RESERVED FOR USE BY THE OPERATING SYSTEM TO STORE DATA, MAINTAIN STACKS, PASS PARMETERS, ETC. SEGMENTS 3 TO F WOULD BE AVAILABLE TO THE USER. IF NOT USED FOR MEMORY, ONE OR MORE OF THE SEGMENTS COULD BE USED FOR MEMORY-MAPPED I/O. SOME OF THE SEGMENTS COULD BE MISSING ALTOGETHER OR COULD CONTAIN LESS THAN 64K. IT WOULD, OF COURSE, BE UP TO THE USER TO REMEMBER WHERE THE HOLES ARE IN HIS ADDRESS SPACE.

ONE IDEA THAT INTRIGUES ME IS TO HAVE A 64K APL INTERPRETER IN SEGMENT 0, WITH THE OTHER 15 SEGMENTS AVAILABLE AS APL WORKSPACES. (APL IS MY FAVOURITE LANGUAGE.)

IT WOULD BE NICE IF RCA (OR SOME ENTERPRISING SECOND-SOURCER) CAME OUT WITH A ONE-CHIP CMOS IC CONTAINING THE ABOVE CIRCUITRY, WITHOUT THE 4515, BUT INCLUDING A LATCH TO TAKE OFF THE UPPER ADDRESS BYTE. THE WHOLE THING COULD BE DONE IN A SINGLE 40 PIN IC. WHICH WOULD ADD A LOT OF POWER TO THE 1802 SERIES, AND ALSO MAKE LIFE EASIER FOR US HOMEBREWERS.

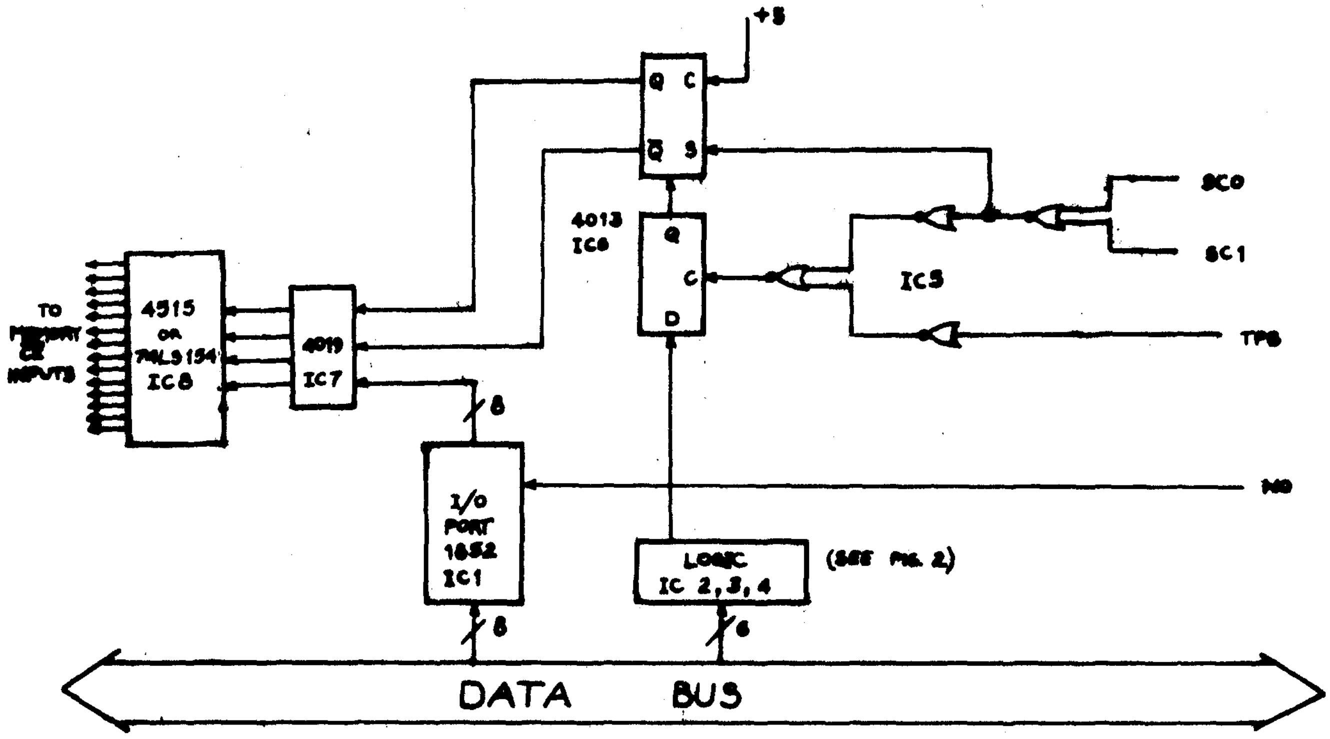


FIGURE 1 - MEMORY MANAGEMENT CIRCUIT

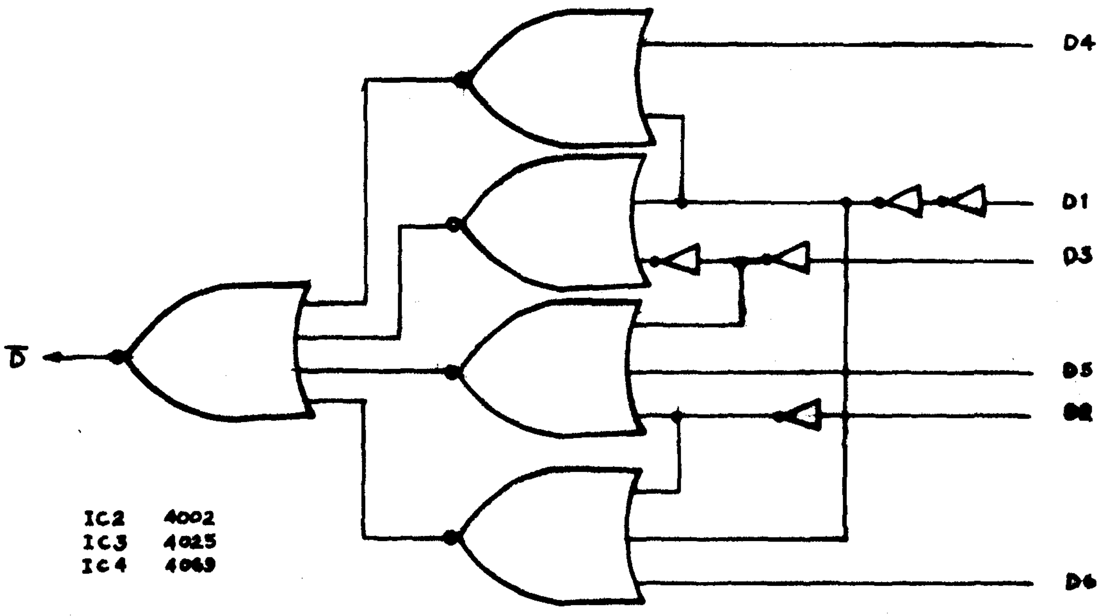


FIGURE 2 - INSTRUCTION / DATA DISCRIMINATOR

CONCLUSION

AS I SAID EARLIER, THIS WHOLE SCHEME IS JUST THEORY AT THE MOMENT. UNFORTUNATELY I AM NOT VERY WELL EQUIPPED FOR THE AMOUNT OF BREADBOARDING THAT THIS SYSTEM WOULD ENTAIL. I WELCOME ANYONE ELSE TO TRY IT OUT AND SEND ME ANY COMMENTS THEY MAY HAVE.

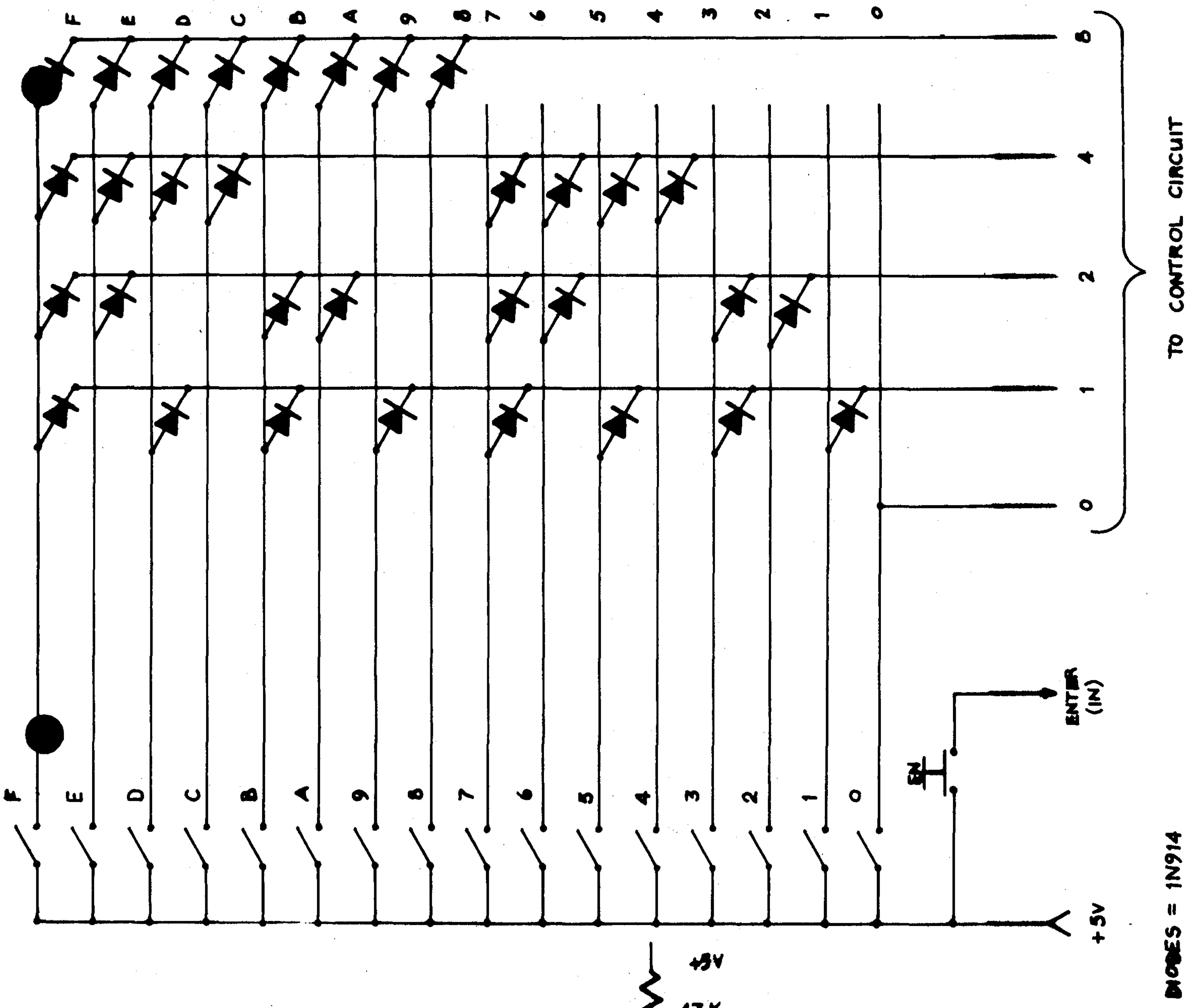
LISTING I - SAMPLE PROGRAM TO COPY DATA BETWEEN SEGMENTS
****

```
* THIS PROGRAM COPIES DATA FROM SEGMENT 1 TO SEGMENT 2.
*
*   R8 - STACK POINTER FOR DATA SEGMENT 1
*   R9 - STACK POINTER FOR DATA SEGMENT 2
*   RA - CONTAINS START ADDRESS OF DATA IN SEGMENT 1
*   RB - CONTAINS DESTINATION ADDRESS IN SEGMENT 2
*   RC - CONTAINS NUMBER OF BYTES TO BE COPIED
*
         SEX     R8
         LDI     =X02      SEGMENT POINTER P=0 D=2
         STR     R8        SAVE ON STACK IN SEGMENT 1
         OUT1              SWITCH DATA SEGMENT TO 2
         DEC     R8
         SEX     R9
         LDI     =X01      SEGMENT POINTER P=0 D=1
         STR     R9        SAVE ON STACK IN SEGMENT 2
         OUT1              SWITCH DATA SEGMENT BACK TO 1
         DEC     R9
LOOP     LDR     RA        GET BYTE TO BE COPIED
         SEX     R8
         OUT1              SWITCH DATA SEGMENT TO 2
         DEC     R8
         STR     RB        STORE IN DATA SEGMENT 2
         SEX     R9
         OUT1              SWITCH DATA SEGMENT TO 1
         DEC     R9
         INC     RA
         INC     RB
         DEC     RC
         GLO     RC
         BNZ     LOOP      BRANCH BACK UNTIL RC IS ZERO
         GHI     RC
         BNZ     LOOP
         *END
```

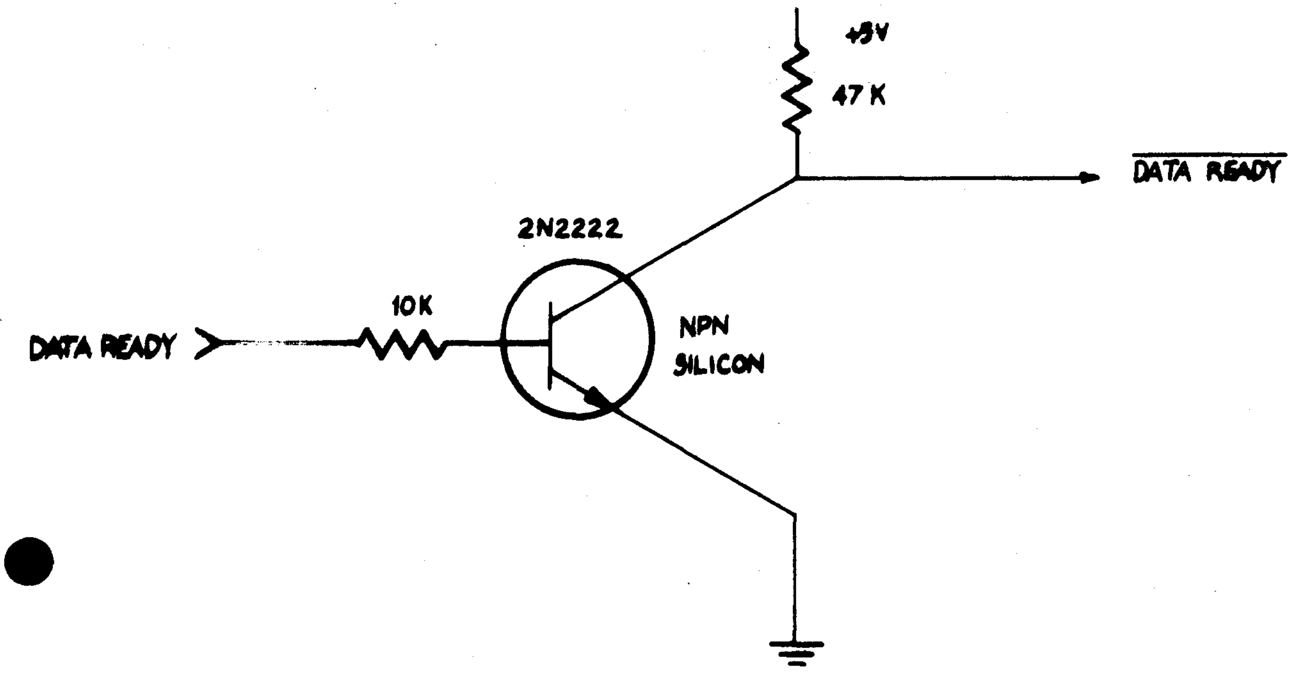
HEXKEYBOARD FOR THE ELF

- by

Here is another simple keyboard that will replace the data switches on the original ELF. The original circuit was taken from ETI Magazine (September 77, pa.60) and modified to hold the data into two latches. The circuit uses TTL chips because they were handy and cost me almost nothing. The circuit could be modified to use CMOS or LSTTL. The keypad comes from a surplus calculator keyboard bought from Rdio Shack. It was modified to make it a 16 spst switches arrangement with one side common. This was done by cutting the printed circuit and rewiring the switches. There are various surplus keyboard on the market and it should be easy to do the same with any keyboard providing you have accessto the switches or the printed circuit. Most keyboard are 'matrix' type so they have to be modified to work in this circuit. 16 spst switches could also be used (push button type). Use IC socket for your circuit. The control circuit was built on a small phenolic board and screwed under the keypad with 12 wires coming out. (8 data, +5v, grd, strobe, enter). The extra switch replaces the IN switch in the original ELF I call it ENTER. A led (kbd ready) indicates the circuit is ready to accept another byte. When the second digit is pressed a data ready strobe is generated and inverted to drive an EF line on the 1802. I am including a short program that will input data in memory sequencely, it uses EF2 for input strobe.



HEX KEYBOARD



DATA READY INVERSION
TO DRIVE EF LINE

1802 TO S-100 BUS CONVERTER

- by David W. Schuler, 3032 Avon Road, Bethlehem, Pa. 18017, U.S.A.

Theory of Operation:

Bus buffer U1 is used to buffer the data lines from the 1802. Since the S-100 bus has separate data in/out lines, the chip select of the buffer is always enabled (pins 1 and 19 = 0). U2 is used to strobe the data from the S-100 bus onto the 1802 bi-directional data bus when a memory read is requested. The chip select of U2 is generated by U3a and U4a on the Elf II only. On the Elf II, pin 1 of U4a is connected to pin 69 of the 86 pin bus on the main board. This line indicates if an on-board memory address has been selected. If pin 69 = 0, the S-100 buffer is disabled. If pin 69 = 1, the S-100 bus buffer is enabled when a memory read request is received.

The Netronics Elf II also requires a latch for the upper 8 address bits (A8 to A15). The required circuit is in Figure 3. This will latch the 8 high order addr. bits, and then the lower 8 address bits. This is because the S-100 board specifications require that the full 16 bit address be present on the bus at the same time.

For both the Quest Super Elf and the Netronics Super Elf, U3a, U3b, and U3c are required to invert the 1802 signals to the required S-100 signals.

Construction:

The entire circuit can be built on one S-100 interface card by either wire wrapping or point-to-point connections. If the Super Elf or the Elf II with Quest Adapter board is used, only a 50 conductor cable and S-100 prototyping card are needed. If only an Elf II is used, a Kludge Card for the 86 pin bus, an S-100 prototyping card and appropriate cable will be required. In either case, a S-100 mother board will be required.

Elf II Note:

In some cases, transistor Q1 on all 4K Static RAM cards will have to be replaced with a higher speed transistor. Also, sometimes diode D4 on the Giant Board will have to be replaced with a higher speed germanium diode.

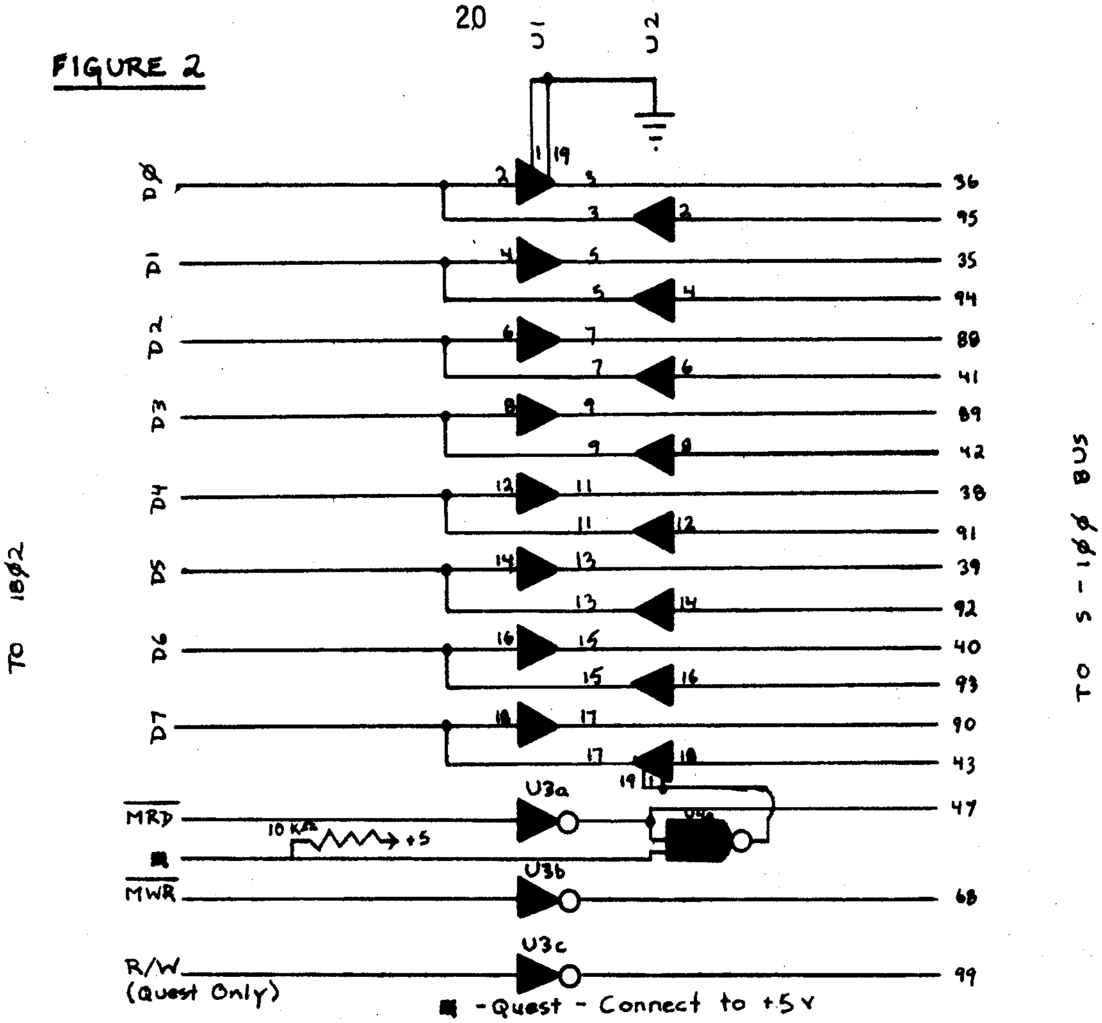
If there are any questions or comments on the interface outlined here, please send a SASE (Self Addressed Stamped Enveloped) along with your questions and I will try to help you out.

FIGURE 1

S-100 (Pin #)	(Signal name)	connect to	1802 (Signal name)
1	+8VDC	"	—
2	+16VDC	"	—
20	GND	"	—
25	pSTVAL	"	$\overline{\text{TPA}}$
29	A5	"	A5
30	A4	"	A4
31	A3	"	A3
32	A15	"	A15 (Note 1)
33	A12	"	A12 (Note 1)
34	A9	"	A9 (Note 1)
35	DO¢	"	DO1 (Note 2)
36	DOØ	"	DOØ (Note 2)
37	A1Ø	"	A1Ø (Note 1)
38	DO4	"	DO4 (Note 2)
39	DO5	"	DO5 (Note 2)
40	DO6	"	DO6 (Note 2)
41	DI2	"	DI2 (Note 2)
42	DI3	"	DI3 (Note 2)
43	DI7	"	DI7 (Note 2)
45	sOUT	"	GND
46	sINP	"	GND
47	sMEMR	"	(Note 3)
50	GND	"	GND
51	+8VDC	"	—
52	-16VDC	"	—
68	MWRT	"	(Note 3)
70	GND	"	—
77	$\overline{\text{pWR}}$	"	$\overline{\text{MWR}}$
79	AØ	"	AØ
80	A1	"	A1
81	A2	"	A2
82	A6	"	A6
83	A7	"	A7
84	A8	"	A8 (Note 1)
85	A13	"	A13 (Note 1)
86	A14	"	A14 (Note 1)
87	A11	"	A11 (Note 1)
88	DO2	"	DO2 (Note 2)
89	DO3	"	DO3 (Note 2)
90	DO7	"	DO7 (Note 2)
91	DI4	"	DI4 (Note 2)
92	DI5	"	DI5 (Note 2)
93	DI6	"	DI6 (Note 2)
94	DI1	"	DI1 (Note 2)
95	DIØ	"	DIØ (Note 2)
99	$\overline{\text{POC}}$	"	(Note 3)
100	GND	"	GND

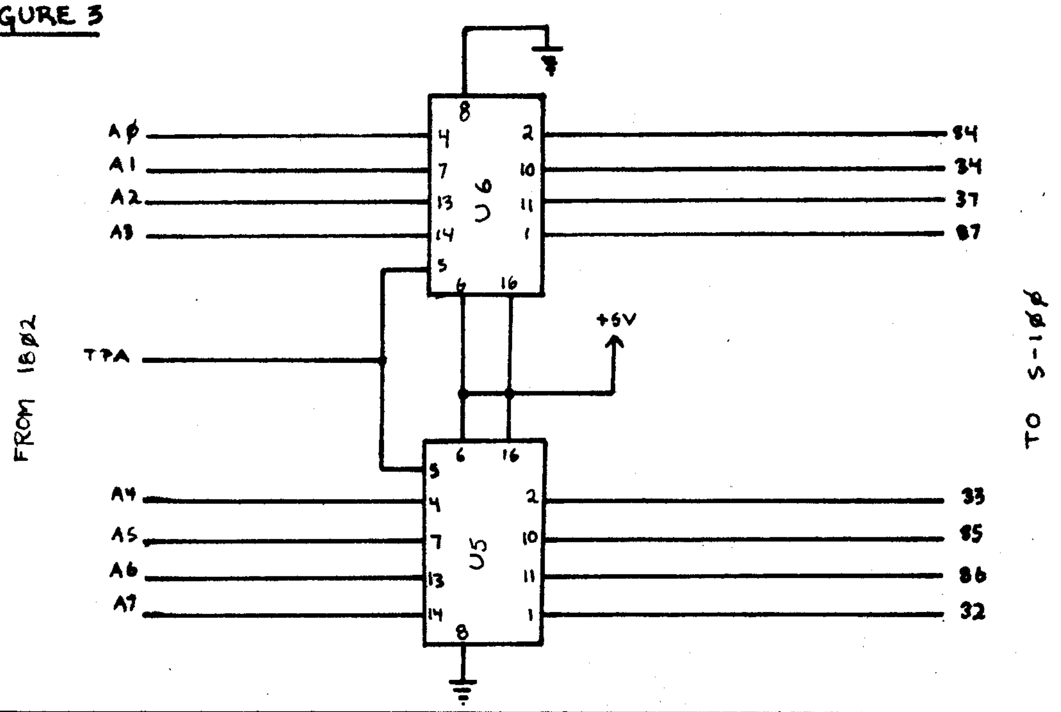
Notes: 1 - Elf II only: See Figure 3 for latch circuit for Address lines A8 to A15.
2 - All: See Figure 2 for data latch.
3 - All: See Figure 2 for circuit data.
4 - Other S-100 lines may have to be added as required by individual boards.

FIGURE 2



■ - Quest - Connect to +5V
Netronics - Pin 69 of 86 pin bus

FIGURE 3



PARTS LIST

U1, U2	-	81LS95 /7
U3	-	74LSØ4
U4	-	74LSØØ
U5, U6	-	4042 or 4508

A NOTABLE ASSEMBLER LOADER

- by D. Stevens, 4 Washington Sq. Village, #13R, N.Y., N.Y., USA, 10012

This routine allows one to assemble long object files without using tape or disk; it can produce an object file with length = RAM/3. I wrote it because I am tired of hand assembling my codes and do not have a terminal or an assembler, just an ELF-II with 8 K RAM. The source (which is machine code + address instructions) is usually less than twice as long as the assembled object file. The routine fits in 3 pages and runs slowly, needing for instance 20 seconds to process its own source (5 pages) and needing 200 seconds to process my operating system (source = 15 pages, object length = 08A0). It requires only 3 pages because it keeps no tables. It will run much faster and can be used as a loader if just the high address bytes are computed.

The source for the assembler has no mnemonics, only op codes and address data. The address data is given in strings of bytes all beginning with 68. For example, the address of a particular op code is given the name "AB" by preceeding the op code with "68 61 AB", it is given the name "CD EF" by preceeding it with "68 62 CD EF". In the source file, a short branch to these locations would be "30 68 11 AB" and "30 68 12 CD EF". When the assembler encounters a data form such as is in the left column below it performs the action described in the right column. Now look at the source listing of the assembler. The first two bytes tell the assembler to write the object code starting at 1000, the 68 03 01 00 tells the assembler that the load address of the object code is 01 00. The next data up to 0417 is op codes, compare with the object code listing. At 0418 is 68 22 01 26, then more op codes. At 0112 in the object listing the byte 03 occurs instead. The name "01 26" is defined at 0823 (the 68 62 01 26) and the address of the following byte is 035D.

Data String	Compiler Action
68 00	A "68" will be inserted in the object code.
68 01 xx yy	The address of the next byte in the object code will be the address of the last byte in the object code plus xx yy. This is a SKIP instruction to the compiler.
68 02 xx yy	The low byte of the address of the next byte in the object code will be yy. The high byte will be the high byte of the last byte plus xx.
68 03 xx yy	The address of the next byte in the object code will be xx yy This is an ORG instruction to to the compiler.

68 1j (j bytes)	The low byte of the address associated with the name will be inserted.
68 2j (j bytes)	The high byte of the address associated with the name will be inserted.
68 3j (j bytes)	The high and low bytes of the address associated with the name will be inserted.
68 4j (j bytes)	A preprocessor changes the source file as follows: if the current object address is on the same page as the address of the name the 4j will be changed to 1j and the byte preceeding the 68 will be changed from xy to 3y. Oherwise the 4j will be changed to 3j and the byte preceeding the 68 will be changed to Cy.
68 6j (j bytes)	The name is associated with the address of the next byte of the object code. The name is a statement label.
68 7j(j bytes)hhll	The name refers to address hhll. This is an EQUATE instruction to the compiler.

The routine is used as follows. It is assumed that your system uses SCRT. The assembler object code is put at 0100-03D8. R8 should point to the low address end of a RAM area, the assembler will store useful data, described below. R9 should point to the start of a source file, and RA should point to the end of the file. The last byte in the file should not be part of a 68... address specifier; if it is, add a 00 to the end of your source. Note the 00 at the end of the source listing of the assembler. The source should start with uu vv 68 03 xx yy, where uu vv is the memory location where the object code will be put by the assembler and xx yy is the load address where the code will run. The assembler output should not overwrite the source. During assembly the high address byte of the source byte being processed is displayed. On return only registers R8, R9, RA and RF are changed. R9 now has uu vv and RA points to the end of the object code just generated. The source is modified; 68 4J patterns are changed to 68 1J or to 68 3J patterns.

If a 68 1j ..... combination occurs and the referenced address is on a different page, the assembled program possibly will not be ok. So a list of all such questionable references is made with R8 the stack pointer. This stack grows upward. If for instance on entry R8 was 1400 and on exit R8 was 1404, then [1400] [1401] is the address in the source of a questionable combination, and also [1402] [1403]. Entries are also made if a 68 3j ..... combination refers to an address which is on the same page.

The routine at 0327 - 0337 puts out the object bytes (the 5F instruction). This may be modified to fill a buffer and write to tape or disk rather than to memory.

If an error in the source data is detected the program branches to C000. This error branch address can be changed, it is at 036C in the object code in the listing and at 0842 in the source part of the listing. The routine at C000 (or wherever) should save R3, R9, and RB. The error table below lists the possible error conditions.

Error Table

R3	Diagnosis
019B	illegal data at RB
01AE	a "4j" in the source was not changed, is the source in ROM?
01B0	illegal data at RB
0227	illegal data at RB-1
0281	RB points to the last byte of a name which wasn't found
02C6	illegal data at R9
0305	illegal data at R9-1

Summary of Usage

1. Set up the source file
   a. First five bytes are uu vv 68 xx yy
   b. No mnemonics
   c. Final byte not compiler instruction (68......)
2. Put the assembler at 0100
3. Put an error handling routine at C000 (or an address you pick)
4. Set up R8, R9 and RA
5. Do SCRT call to 0100
6. If the routine exits to the error routine use the table above
7. If the routine makes a normal exit, check R8 and the data in the R8 stack.

Listing of Assembler Object Code

```
0100  E2 8E 73 9E  73 87 73 97  73 82 FF 24  A7 92 7F 00
0110  B7 F8 03 BE  F8 5D AE DE  04 19 19 D4  03 38 05 40
0120  F0 68 FF 02  D4 03 8C 17  07 32 69 8A  73 9A 73 17
0130  47 BA BB BC  07 AA AB AC  1B 1C D4 02  A2 47 BD 12
0140  42 BA 02 AA  D4 02 4A E7  9D F3 17 32  5B 0C FB 70
0150  5C 2C 2C 0C  FA 0F F9 C0  5C 30 1B 0C  FB 50 5C 2C
0160  2C 0C FA 0F  F9 30 5C 30  1B 99 BB 89  AB 2B 19 19
0170  19 E9 0B F7  AC 2B 29 0B  77 BC 1B 29  29 8B 52 E2
0180  8A F3 3A 8A  9B 52 9A F3  32 FE 1B 9B  52 64 22 0B
0190  FB 68 3A BE  1B 0B FE 3B  9B DE 00 F6  F6 F6 F6 F6
01A0  FE FC A4 A3  30 C5 30 C8  30 DC 30 E2  DE 00 DE 00
01B0  30 F9 0B FA  0F FC 02 1B  FF 01 3A B7  30 7D 0B B5
01C0  D4 03 27 30  7D C0 02 16  9B 58 18 8B  58 28 D4 02
01D0  4A 9D E7 F3  32 D8 18 18  17 07 30 BF  D4 02 4A 47
01E0  30 BF 9B 58  18 8B 58 28  D4 02 4A 9D  E7 F3 3A F2
01F0  18 18 47 B5  D4 03 27 30  D9 0B FA 0F  30 B7 9F BA
0200  8F AA DE 08  29 09 AF 29  09 B9 8F A9  12 42 B7 42
0210  A7 42 BE 02  AE D5 4B 32  27 FF 01 32  2D FF 01 32
0220  3D FF 01 32  46 DE 00 F8  68 2B C0 01  BF 1B 0B 52
0230  8D F4 AD 2B  4B 52 9D 74  BD 2D C0 01  7D 4B 52 9D
0240  F4 BD 0B AD  30 3A 4B BD  30 42 DE 01  0B FA 0F A8
```

```
0250  B8 1B FF 01  3A 51 9B BF  8B AF OF 2F  E7 73 F8 FF
0260  73 28 88 3A  5A 98 F9 60  73 F8 EF 73  F8 68 73 F8
0270  FF 73 98 FC  02 73 D4 03  8C 17 47 FB  55 32 81 DE
0280  00 47 BA 07  AA 1A 0A FA  10 3A 92 2A  D4 02 A2 DE
0290  02 D5 98 1A  FF 01 3A 93  1A 1A E7 0A  73 2A 0A 57
02A0  30 8F DE 01  89 52 8A F3  3A B8 99 52  9A F3 3A B8
02B0  88 57 27 98  57 DE 02 D5  49 FB 68 32  C0 18 30 A4
02C0  09 FE 3B C6  DE 00 F6 F6  F6 F6 F6 FE  FC CF A3 30
02D0  F0 30 ED 30  ED 30 EC 30  EC DE 00 30  E9 49 19 19
02E0  FA OF 19 FF  01 3A E2 30  A4 49 30 E0  18 18 30 E9
02F0  49 C2 03 05  FF 01 C2 03  09 FF 01 C2  03 19 FF 01
0300  C2 03 23 DE  00 18 C0 02  A4 19 09 52  88 F4 A8 29
0310  49 19 52 98  74 B8 28 30  20 49 52 98  F4 B8 49 A8
0320  C0 02 A4 49  B8 30 1E 8D  52 8C F4 AF  9D 52 9C 74
0330  BF 95 5F 94  B5 1D 9F D5  46 AF E7 46  73 2F 8F 3A
0340  3B D5 8D 73  9D 73 8C 73  9C 73 8B 73  9B 73 30 5C
0350  8A 73 9A 73  89 73 99 73  88 73 98 73  D3 43 E2 F6
0360  33 50 F6 33  6E F6 33 42  F6 33 7D C0  C0 00 12 42
0370  B8 42 A8 42  B9 42 A9 42  BA 02 AA 30  5C 12 42 BB
0380  42 AB 42 BC  42 AC 42 BD  02 AD 30 5C  DE 01 DE 04
0390  17 07 BB AB  97 B8 87 A8  1A 2B 2A 8B  3A 99 9B AB
03A0  E7 99 BC 89  AC 98 B7 88  A7 4C 17 F2  17 F3 3A C1
03B0  2B 8B 3A A9  89 73 99 73  F8 55 57 27  DE 08 DE 02
03C0  D5 E2 99 52  9A F3 32 CB  19 30 9E 89  52 8A F3 3A
03D0  C8 2B 8B 32  BA 17 17 30  D1

0400  11 00 68 03  01 00 E2 8E  73 9E 73 87  73 97 73 82
0410  FF 24 A7 92  7F 00 B7 F8  68 22 01 26  BE F8 68 12
0420  01 26 AE 68  62 00 E0 DE  04 19 19 68  62 00 E1 D4
0430  68 32 01 20  05 40 F0 68  00 FF 02 D4  68 32 00 D0
0440  17 07 32 68  12 00 E3 8A  73 9A 73 17  47 BA BB BC
0450  07 AA AB AC  1B 1C D4 68  32 01 0A 47  BD 12 42 BA
0460  02 AA D4 68  32 01 00 E7  9D F3 17 32  68 12 00 E2
0470  0C FB 70 5C  2C 2C 0C FA  0F F9 C0 5C  30 68 12 00
0480  E1 68 62 00  E2 0C FB 50  5C 2C 2C 0C  FA 0F F9 30
0490  5C 30 68 12  00 E1 68 62  00 E3 99 BB  89 AB 2B 19
04A0  19 19 E9 0B  F7 AC 2B 29  0B 77 BC 1B  29 29 68 62
04B0  00 E4 8B 52  E2 8A F3 3A  68 12 00 E5  9B 52 9A F3
04C0  32 68 12 00  F3 68 62 00  E5 1B 9B 52  64 22 0B FB
04D0  68 00 3A 68  12 00 E9 1B  0B FE 3B 68  12 00 E6 DE
04E0  00 68 62 00  E6 F6 F6 F6  F6 F6 FE FC  68 12 00 E7
04F0  A3 68 62 00  E7 30 68 12  00 EB 30 68  12 00 EC 30
0500  68 12 00 EF  30 68 12 00  F0 DE 00 DE  00 30 68 12
0510  00 F2 0B FA  0F FC 02 68  62 00 E8 1B  FF 01 3A 68
0520  12 00 E8 30  68 12 00 E4  68 62 00 E9  0B 68 62 00
0530  EA B5 D4 68  32 01 1F 30  68 12 00 E4  68 62 00 EB
0540  C0 68 32 00  F4 68 62 00  EC 9B 58 18  8B 58 28 D4
0550  68 32 01 00  9D E7 F3 32  68 12 00 ED  18 18 68 62
0560  00 ED 17 68  62 00 EE 07  30 68 12 00  EA 68 62 00
0570  EF D4 68 32  01 00 47 30  68 12 00 EA  68 62 00 F0
0580  9B 58 18 8B  58 28 D4 68  32 01 00 9D  E7 F3 3A 68
0590  12 00 F1 18  18 68 62 00  F1 47 B5 D4  68 32 01 1F
05A0  30 68 12 00  EE 68 62 00  F2 0B FA 0F  30 68 12 00
05B0  E8 68 62 00  F3 9F BA 8F  AA DE 08 29  09 AF 29 09
```

```
05C0  B9 8F A9 12  42 B7 42 A7  42 BE 02 AE  D5 68 62 00
05D0  F4 4B 32 68  12 00 F5 FF  01 32 68 12  00 F6 FF 01
05E0  32 68 12 00  F8 FF 01 32  68 12 00 FA  DE 00 68 62
05F0  00 F5 F8 68  00 2B C0 68  32 00 EA 68  62 00 F6 1B
0600  0B 52 8D F4  AD 2B 4B 52  9D 74 BD 2D  68 62 00 F7
0610  C0 68 32 00  E4 68 62 00  F8 4B 52 9D  F4 BD 68 62
0620  00 F9 0B AD  30 68 12 00  F7 68 62 00  FA 4B BD 30
0630  68 12 00 F9  68 62 01 00  DE 01 0B FA  0F A8 B8 68
0640  62 01 01 1B  FF 01 3A 68  12 01 01 9B  BF 8B AF 68
0650  62 01 02 0F  2F E7 73 F8  FF 73 28 88  3A 68 12 01
0660  02 98 F9 60  73 F8 EF 73  F8 68 00 73  F8 FF 73 98
0670  FC 02 73 D4  68 32 00 D0  17 47 FB 55  32 68 12 01
0680  03 DE 00 68  62 01 03 47  BA 07 AA 1A  0A FA 10 3A
0690  68 12 01 05  2A D4 68 32  01 0A 68 62  01 04 DE 02
06A0  D5 68 62 01  05 98 68 62  01 06 1A FF  01 3A 68 12
06B0  01 06 1A 1A  E7 0A 73 2A  0A 57 30 68  12 01 04 68
06C0  62 01 0A DE  01 68 62 01  0B 89 52 8A  F3 3A 68 12
06D0  01 0C 99 52  9A F3 3A 68  12 01 0C 88  57 27 98 57
06E0  DE 02 D5 68  62 01 0C 49  FB 68 00 32  68 12 01 0D
06F0  18 30 68 12  01 0B 68 62  01 0D 09 FE  3B 68 12 01
0700  0E DE 00 68  62 01 0E F6  F6 F6 F6 F6  FE FC 68 12
0710  01 0F A3 68  62 01 0F 30  68 12 01 15  30 68 12 01
0720  14 30 68 12  01 14 30 68  12 01 13 30  68 12 01 13
0730  DE 00 30 68  12 01 12 49  19 19 68 62  01 10 FA 0F
0740  68 62 01 11  19 FF 01 3A  68 12 01 11  30 68 12 01
0750  0B 68 62 01  12 49 30 68  12 01 10 68  62 01 13 18
0760  68 62 01 14  18 30 68 12  01 12 68 62  01 15 49 C2
0770  68 32 01 16  FF 01 C2 68  32 01 17 FF  01 C2 68 32
0780  01 18 FF 01  C2 68 32 01  1B DE 00 68  62 01 16 18
0790  C0 68 32 01  0B 68 62 01  17 19 09 52  88 F4 A8 29
07A0  49 19 52 98  74 B8 28 30  68 12 01 1A  68 62 01 18
07B0  49 52 98 F4  B8 68 62 01  19 49 A8 68  62 01 1A C0
07C0  68 32 01 0B  68 62 01 1B  49 B8 30 68  12 01 19 68
07D0  62 01 1F 8D  52 8C F4 AF  9D 52 9C 74  BF 95 5F 94
07E0  B5 1D 9F D5  68 62 01 20  46 AF E7 68  62 01 21 46
07F0  73 2F 8F 3A  68 12 01 21  D5 68 62 01  23 8D 73 9D
0800  73 8C 73 9C  73 8B 73 9B  73 30 68 12  01 25 68 62
0810  01 24 8A 73  9A 73 89 73  99 73 88 73  98 73 68 62
0820  01 25 D3 68  62 01 26 43  E2 F6 33 68  12 01 24 F6
0830  33 68 12 01  27 F6 33 68  12 01 23 F6  33 68 12 01
0840  28 C0 C0 00  68 62 01 27  12 42 B8 42  A8 42 B9 42
0850  A9 42 BA 02  AA 30 68 12  01 25 68 62  01 28 12 42
0860  BB 42 AB 42  BC 42 AC 42  BD 02 AD 30  68 12 01 25
0870  68 62 00 D0  DE 01 DE 04  17 07 68 62  00 D1 BB AB
0880  97 B8 87 A8  1A 68 62 00  D2 2B 2A 8B  3A 68 12 00
0890  D2 68 62 00  D3 9B AB E7  99 BC 89 AC  98 B7 88 A7
08A0  68 62 00 D4  4C 17 F2 17  F3 3A 68 12  00 D6 2B 8B
08B0  3A 68 12 00  D4 89 73 99  73 F8 55 68  62 00 D5 57
08C0  27 DE 08 DE  02 D5 68 62  00 D6 E2 99  52 9A F3 32
08D0  68 12 00 D8  68 62 00 D7  19 30 68 12  00 D3 68 62
08E0  00 D8 89 52  8A F3 3A 68  12 00 D7 68  62 00 D9 2B
08F0  8B 32 68 12  00 D5 17 17  30 68 12 00  D9 00
```

A HARDWARE CLOCK FOR THE 1802

- by J. Swofford, 2302 N. Fairview Ave. Decater, Illinois, USA, 62526

Last spring I ordered an OKI real-time clock/calendar to fill a need for time-keeping in my ELF II. My circuit built around this I.C. is memory-mapped via 8255 PPI and allows access to time in hours, minutes and seconds and the date as well as the day-of-week. Time can be kept in either a 12 or 24 hour format and leap year compensation is provided. My system has the 8255 located at FF00H to FF03H. Unfortunately, the Netronics monitor interferes with locations beyond F0FFH, so some ELF II owners may wish to locate the 8255 elsewhere or, as I did, enable the Netronics monitor only for F000H to F0FFH (as it should have been, anyway).

INTERFACING THE CLOCK

The data sheet from OKI shows a suggested arrangement using the Intel 8255 PPI. Since I already had a few spares, it seemed the easiest route to take. The capabilities of the PPI are too extensive to cover here so I will be concerned with only those aspects which affect this application. Memory locations FF00H to FF03H will be assumed. All three of the 8255 I/O ports are used. The PPI operates here in mode 0 in two configurations; one to read from the clock and one to write to the clock (see figure 1). These states are created by writing 80H into location FF03H for clock WRITE and 90H for clock READ. These bytes control how the 8255 itself operates. An 80H will allow port A to be an output port (ports B and C remain output ports for both clock READ and clock WRITE). Similarly, 90H allows port A to input the data nibble from the clock data lines. Port B is used to provide the clock address nibble and port C is for clock control (see table 1). When the 8255 is reset, all 24 bits (three 8 bit ports) normally float. In this circuit, R1-R16 will pull these lines high. This would normally place the clock in a HOLD state, stopping the time increment. Therefore, PC0 (bit 0-port C) is inverted before going to the chip select pin on the clock. This removes all control (including HOLD) from the

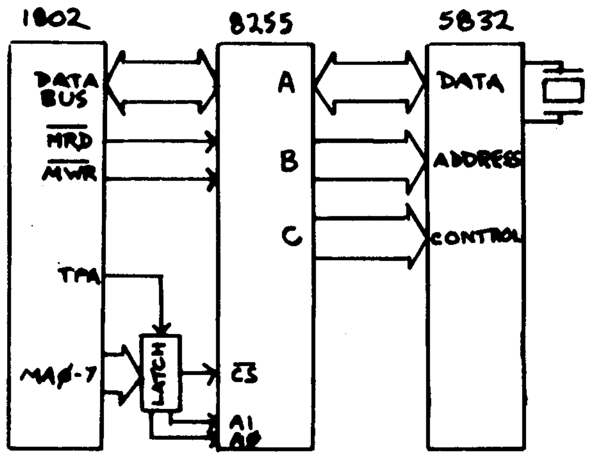


FIGURE 1

clock and allows it to continue keeping time. The upper portions of ports A and B (PA4-7 and PB4-7) are not used. PA4-7 is tied to ground through 10K resistors to avoid having to 'strip' them off in software during clock READ.

<u>ADDRESSING</u>

Since I use other 8255s in my system, the schematic shows a 74LS154 providing $\overline{CS}$ for the clock's 8255. Using this arrangement, 16 different PPIs can be selected yielding 48 8-bit I/O ports. Many good methods of high order address latching and chip selection have appeared in Ipso Facto. This one works for me. Since you may not have an interest in this expansion idea, you may wish to use an alternate method of chip select which eliminates the 74LS30, 74LS32, 74LS154 and 74LS374 (figure 2). Cut the trace on the Netronics Giant Board between pin 8 on A13 and pin 16 on A10. Run a line from A13 pin 8 to point Ⓐ (chip select-see clock schematic). A10 pin 16 should be tied high to disable the ROM. The 8255 will now have addresses F000H to F003H. The Demo program will run unmodified with this arrangement. I have tried this-it works. It should be possible to place a switch on the Giant Board to switch from the monitor to the clock and vice versa. If you are using Netronics' Full BASIC, the only way to into BASIC is through the monitor. You may want to develop the idea in figure 3 using SF1# to enable the clock once BASIC has been entered (I have not tried this). The modular appearance of the address and select circuitry is to emphasize the idea of using presently available circuitry you may already have in your system. If you have the upper address bits already latched from some other project, try connecting those lines to the 74LS30 at point Ⓑ, eliminating the 74LS374. I should mention that the Netronics 4K memory board has these address bits already latched and marked on the board itself. I have not tried using them, however. Note that CMOS was not used. I have had no problems using TTL but you may wish to use the 74C series, anyway.

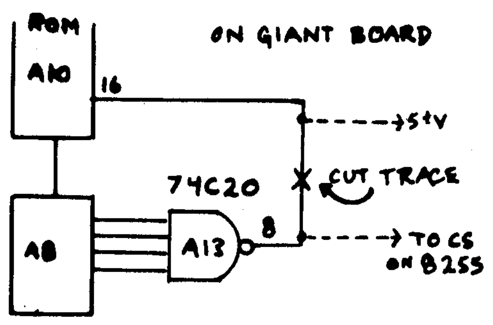


FIGURE 2

THE CLOCK

The MSM5832 is basically a digital watch in a DIP package. Each of the 13 digits available (HH MM SS W MM DD YY) must be called for one at a time. This is done by addressing the digit to be read/written (don't confuse with the ELF address/data lines), commanding the clock and reading/writing the clock (see figure 4). The HOLD line should not be high for more than one second in order that the clock can increment. The demo program does not use HOLD on READ due partly to this requirement (see figure 5).

The H10 and D10 digits are special. The H10 digit contains flags for PM and 24 hour operation. The D10 digit contains a flag for leap year (table 2). The ±30 second adjust causes 1 minute to be added to the LSB of the minutes if seconds are 30 or more as the seconds are set to zero. If less than 30 seconds, only the seconds are affected. Battery backup is provided by 2 alkaline penlight cells. Backup is not required; it is recommended. It is a pain to reload the clock every time the computer has been powered down. Just ask any TRS-80 model III owner. The very low drain of the 5832 should give many months (years?) of service. Time regulation is through C1. Decrease C1 to speed the clock; increase C1 to slow the clock.

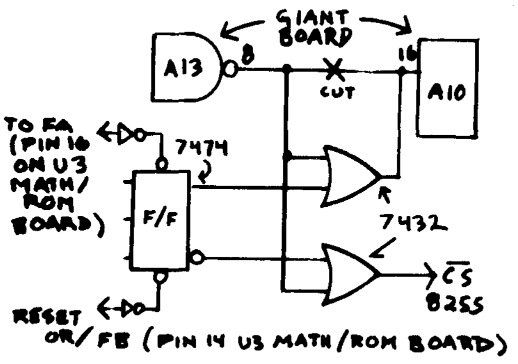

GIANT BOARD
A13
8
CUT
16
A10
TO FA (PIN 16 ON U3 MATH/ROM BOARD)
7474
F/F
7432
$\overline{CS}$ 8255
RESET OR/FB (PIN 14 U3 MATH/ROM BOARD)

FIGURE 3

SOFTWARE

The clock demo was written for an ELF II using the Netronics video board and the 8255 at FF00H. In Cenker's BASIC ver. 5, this program uses about 1.5K of memory without REMs. The program is easily altered to run at any location by changing line 60 ("#" and "@" indicates hexidecimal in Quest BASIC). The clock READ did not use HOLD since BASIC tied up the clock too long and caused it to lose time. Constant interrogation of the clock should be done in machine code. The time and date is updated on the CRT about every 4 seconds. A more frequent update can be achieved by leaving out the unnecessary statements and not calling for the date everytime.

COMMENTS

Information on obtaining parts is in order. The MSM5832 and crystal (32.768 Hz) is available form several sources including Concord Computer Products, 1971 So. State College, Anaheim, Ca. 92806 and Digi-Key, Hiway 32 South, P.O. Box 677, Thief River Falls, Mn. 56701. Both suppliers provide the data sheet. The 8255 and other ICs are available from just about anywhere. The trimmer (C1) is sold by Jameco, 1355 Shoreway Road, Belmont, Ca. 94002. The components can be mounted on a Radio Shack perf board P/N 276-1395. For those of us who aren't made of money, here are a couple of suggestions. Don't throw away that digital watch. You may be able to use the small crystal in it like I did (32.768Hz). As for the other parts, try a hamfest. I have saved millions(?) by doing my parts procurement at hamfests. Ask any amateur radio operator for information on where and when.

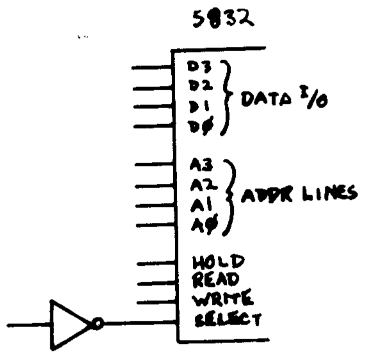


FIGURE 4

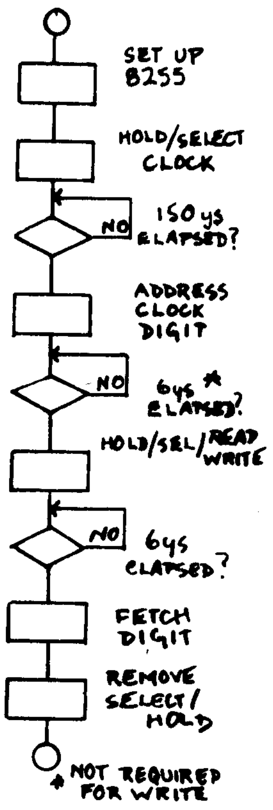


FIGURE 5

One aspect of the 5832 which I have not mentioned is the interrupt signals (figure 6). While I have not made use of them in this circuit, there are many ways to utilize these pulses which, I hope, others will find. For example, the 1024 Hz and 1 Hz could be combined for a software controlled beeper (see figure 7) while the clock is not being accessed. The 60 Hz pulse could be used for a time base in an UPS (Uninterruptable Power Supply) system. The 1 Hz could be used to flash lights. I suppose you could even use these signals for interrupts. What about the clock? How about this:

*Interrupt driven timer
*Countdown timer with a HOLD capability
*Event control (BSR control system)
*Data timestamping

The design is relatively simple, the applications are many. From control to display, this clock can be a useful addition to any system.

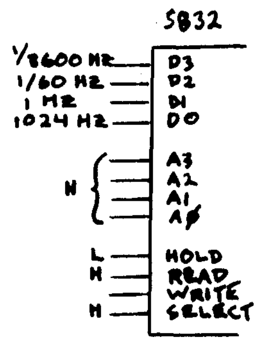
Ref. OKI data sheet, March 1980
Intel data manual, October 1977, pp 6-223 to 6-240

TABLE 1 demo program variables summary

term	hex value	decimal value	used in program for:
A	FF00	65280	ELF memory address-port A (clk I/O)
B	FF01	65281	ELF memory address-port B (clk adr)
C	FF02	65282	ELF memory address-port C (clk contl)
D	FF03	65283	ELF memory address-8255 contl port
E	90	144	control for 8255 (A=input)
F	20	32	port C-enable clk SELECT/READ
G	50	80	port C-enable clk SELECT/HOLD/WRITE
H	80	128	8255 control-ports A,B,C are outputs

TABLE 2 clock functions

clock data		clock address (MSB LSB)	comments
seconds	lsb	0000	seconds are automatically set to zero
seconds	msb	0001	when clk is written into
minutes	lsb	0010	
minutes	msb	0011	
hours	lsb	0100	
hours	msb	0101	bit 2 high=PM bit 3 high=24 hour format
week		0110	range 0-6 (0=Sunday)
day	lsb	0111	
day	msb	1000	bit 2=leap year
month	lsb	1001	
month	msb	1010	
year	lsb	1011	
year	msb	1100	



HOLD MUST BE LOW,
READ, SELECT AØ-7 MUST BE HIGH
TO ALLOW INT. PULSES

FIGURE 6

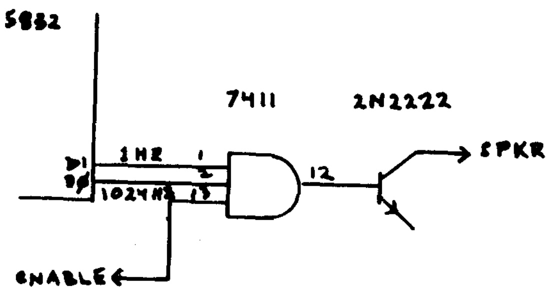
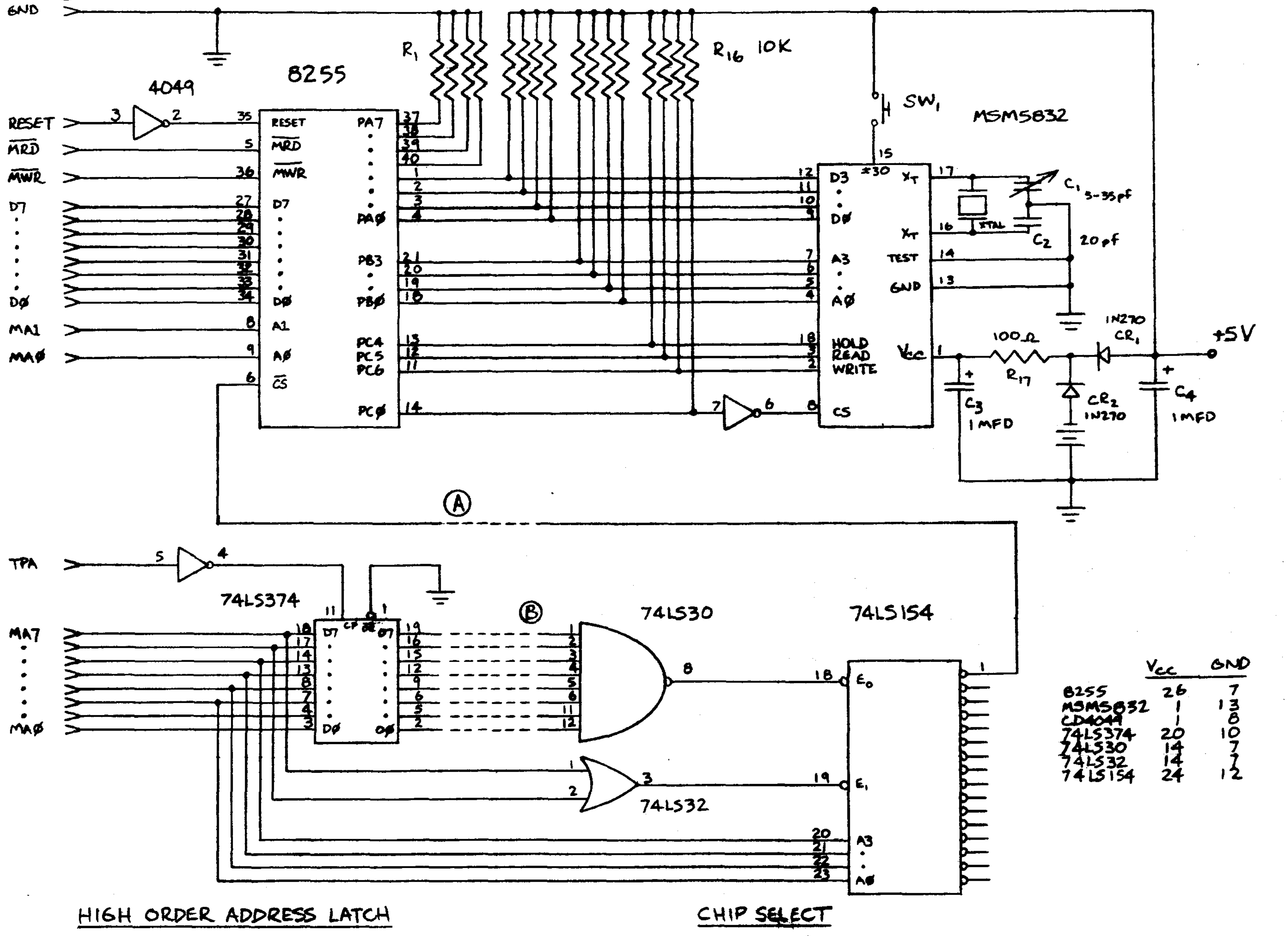


FIGURE 7

```
10  REM*************************
20  REM****   CLOCK DEMO   *****
30  REM** J. SWOFFORD 11-7-81 **
40  REM*************************
50  REM*** SET MEMORY POINTERS *
60  A=@FF00:B=@FF01:C=@FF02:D=@FF03
70  REM*** DATA USED FOR CONTROL
80  E=#90:H=#80:REM 8255 CONTROL BYTES
90  F=#20:G=#50:REM CLOCK CONTROL BYTES
100 DIM T(13)
110 T$=" ":REM INITIALIZE T$
120 B$="N":REM INITIALIZE B$
130 REM
140 REM***  INPUT TIME/DATE  ***
150 CLS
160 INPUT "CHANGE TIME/DATE--Y OR N" A$
170 IF A$="N" GOTO 450
180 IF A$<>"Y" GOTO 160
190 PRINT "INPUT TIME IN THIS FORMAT"
200 PRINT" H,H,M,M"
210 INPUT T(6),T(5),T(4),T(3)
220 T(2)=0:T(1)=0:REM SECONDS ARE SET TO ZERO BY CLK, ANYWAY
230 INPUT "24 HOUR FORMAT--Y/N" B$
240 IF B$="Y" T(6)=T(6)+#08:GOTO 270:REM BIT 3 INDICATES 24 HR FORMAT
250 INPUT "(A)M OR (P)M" C$
260 IF C$="P" T(6)=T(6)+#04:REM SET BIT 2 FOR PM
270 REM
280 REM
290 PRINT "INPUT DATE IN THIS FORMAT"
300 PRINT" W,M,M,D,D,Y,Y"
310 INPUT T(7),T(11),T(10),T(9),T(8),T(13),T(12)
320 T(7)=T(7)-1:REM RANGE FOR DAY-OF-WEEK IS 0-6 (SUNDAY-SATURDAY)
330 INPUT "LEAP YEAR--Y OR N" D$
340 IF D$="Y" T(9)=T(9)+#04:REM BIT 2 FOR LEAP YEAR
350 REM
360 INPUT "PRESS RETURN TO ENTER TIME/DATE" E$:REM ENTER TIME ON QUEUE
370 REM
```

```
380 POKE (D,H):REM SET UP 8255-PORTS A, B, C AS OUTPUTS
390 POKE (C,G):REM PORT C (CLK CONTROL) CLOCK WRITE
400 REM
410 FOR I=1 TO 13
420 POKE (B,I-1):REM SET CLK ADDRESS LINES
430 POKE (A,T(I)):REM SHOVE T(I) INTO CLOCK
440 NEXT I
450 POKE (D,E):REM MAKE PORT A AN INPUT, PORTS B & C ARE OUTPUTS
460 POKE (C,F):REM CLOCK READ
470 REM
480 CLS
490 FOR I=1 TO 7
500 READ A$(I):REM USED FOR DAY-OF-WEEK
510 NEXT I
520 REM
530 PRINT CHR$(4);:REM CURSOR HOME
540 REM
550 FOR I=1 TO 13
560 POKE (B,I-1):REM CLOCK ADDRESS LINES
570 T(I)=PEEK(A):REM LOAD T(I)
580 NEXT I
590 T$="  AM":REM T$ DEFAULT
600 IF INT(T(6)/#02)>0 T$="  PM":REM IF BIT 1 THEN PM
610 IF INT(T(6)/#08)>0 T$=" HRS":REM IF BIT 3 THEN 24 HR FORMAT
620 IF T(6)>2 IF T(6)<7 T(6)=T(6)-#04:REM STRIP OF BIT 2
630 IF T(6)>7 T(6)=T(6)-#08:REM STRIP OFF BIT 3
640 IF T(9)>3 T(9)=T(9)-#04:REM STRIP OFF BIT 2
650 PRINT "TIME  ";T(6);T(5);":";T(4);T(3);":";T(2);T(1);
660 PRINT T$
670 PRINT A$(T(7)+1);
680 PRINT TAB(10);T(11);T(10);"/";T(9);T(8);"/";T(13);T(12)
690 GOTO 530:REM GO UPDATE TIME AND DATE FOR DISPLAY
700 DATA "SUNDAY","MONDAY","TUESDAY","WEDNESDAY","THURSDAY"
710 DATA "FRIDAY","SATURDAY"
720 END
```



	Vcc	GND
8255	26	7
MSM5832	1	13
CD4049	1	8
74LS374	20	10
74LS30	14	7
74LS32	14	7
74LS154	24	12

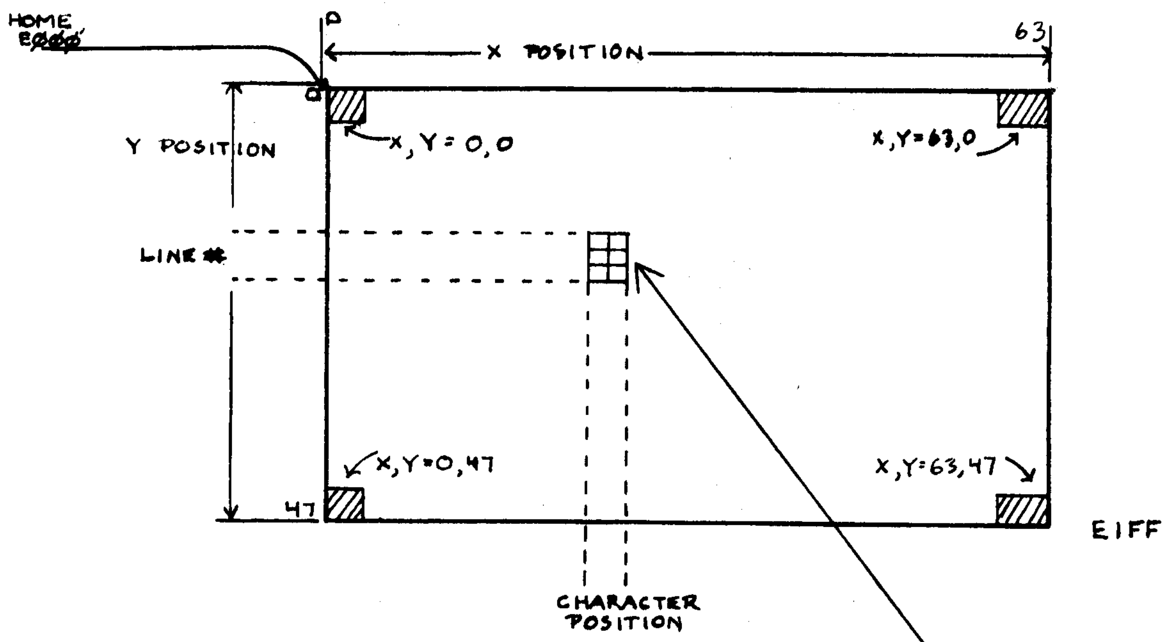
1802 REAL-TIME CLOCK / CALENDAR

```
                 ***********************   *************************************
                 ****************************************************************
                 *                                                              *
                 *         POINT PLOTTER - 6847 SEMIGRAPHIC-SIX MODE (64 X 48)   *
                 *                                                              *
                 *         CREATED:  9 NOV 1981   REV. 0.02   13 NOV 81         *
                 *                                                              *
                 ****************************************************************
                 *                                                              *
                 *         ENTRY:  PROGRAM LABEL IS -- JUMPLOT    (SOURCE)      *
                 *                 OBJCODE  "     " -- PLOTOUT                  *
                 *                 ASM. LISTING   " -- PLOTLIST                 *
                 *         AUTHOR: JORGEN MUNCK                                 *
                 *                                                              *
                 *         DESCRIPTION:  THESE ROUTINES WERE WRITTEN TO BE USED *
                 *                 WITH TOM PITTMAN'S TINY BASIC USR FUNCTION.  *
                 *                 ...C = USR(4195)............CLEAR SCREEN     *
                 *                 ...P = USR(4096,X,Y)........PLOT POINT       *
                 *                    WHERE:  X = 0 TO +63, Y = 0 TO 47         *
                 *                 SCREEN LOCATION = 32*Y/3 + X/2 + E000H (HEX) *
                 *                                                              *
                 ****************************************************************
                 ****************************************************************

                 *-------  EQUATES  -------------------------------------------*

0000             ZEROS     EQU     0

                 *-------------------------------------------------------------*

0000                       ORG     1000H
1000 8A          INITLZ    GLO     A           D <-- ARG3(Y), RA.1 <-- Y;
1001 BA                    PHI     A             0 - 47 PERMISSIBLE
1002 F800                  LDI     ZEROS       CLEAR OUT RA.0 FOR CT OF Y/3
1004 AA                    PLO     A
1005 FE                    SHL                 SHIFT '0' INTO DATA FLAG
1006 9A          YCOOR     GHI     A           GET Y INTO ACCUMULATOR
1007 FF03        SUBT      SMI     #03         SUBTRACT 3; DF = 0 IF BORROW
1009 3B0E                  BNF     QUOTY       IF BORROW, THEN QUOTIENT DONE
100B 1A                    INC     A             ELSE, INCR REG A FOR Y/3
100C 3007                  BR      SUBT          QUOTIENT COUNT
100E FC03        QUOTY     ADI     #03         FIX FOR LAST SUBTRACT
1010 B8                    PHI     8             AND STORE IN R8.1
1011 8A          MULT      GLO     A           GET Y/3 FROM RA.0
1012 FE                    SHL                   MULTIPLY
1013 FE                    SHL                     BY 32
1014 FE                    SHL
1015 FE                    SHL
1016 FE                    SHL
1017 AA                    PLO     A           STORE (32 X Y/3) IN RA.0
1018 F8E0                  LDI     #E0         SCREEN HOME AT #E000
101A 3B1E                  BNF     HOMEHI      IF DF = 0, 32 X Y/3 IS < 9 BITS
101C 7C00                  ADCI    ZEROS         ADD OVERFLOW TO LSB OF RA.1
101E BA          HOMEHI    PHI     A           SAVE CURSOR HIGH PORTION
101F 88          XCOOR     GLO     8           GET X (ARG 2); 0-63 PERMISSIBLE
```

```
1020 F6     QUOTX    SHR            PUSH REMAINDER OF X INTO DF
1021 52              STR    2         SAVE X/2
1022 F800            LDI    ZEROS   CLEAR ACCUMULATOR
1024 7E              SHLC             PUSH DF INTO ACCUM
1025 A8              PLO    8         AND SAVE REMAINDER OF X/2
1026 8A              GLO    A       GET 32 X Y/3
1027 F1              OR               COMBINE WITH X/2
1028 AA     SCREEN   PLO    A       SCREEN ADDRESS COMPLETE
1029 98     CHROW    GHI    8       GET Y/3 REMAINDER (ROW)
102A FE              SHL              MULTIPLY BY 2
102B B8              PHI    8         SAVE
102C 88     CHCOL    GLO    8       GET REMAINDER FROM X/2
102D 76              SHRC             DF = 1 IF REMAINDER WAS A ONE
102E 98              GHI    8       GET ROW INFO INTO ACCUM
102F 7C00            ADCI   ZEROS     ADD CARRY
1031 3247            BZ     SIX     TEST FOR POSITION OF PIXEL
1033 FB01            XRI    #01
1035 324A            BZ     FIVE
1037 FB03            XRI    #03
1039 324D            BZ     FOUR
103B FB01            XRI    #01
103D 3250            BZ     THREE
103F FB07            XRI    #07
1041 3253            BZ     TWO
1043 FB01            XRI    #01
1045 3256            BZ     ONE
1047 F820   SIX      LDI    #20     THESE ARE HEXADECIMAL WEIGHTING
1049 C8              LSKP             FOR PIXEL POSITIONING
104A F810   FIVE     LDI    #10
104C C8              LSKP
104D F808   FOUR     LDI    #08
104F C8              LSKP
1050 F804   THREE    LDI    #04
1052 C8              LSKP
1053 F802   TWO      LDI    #02
1055 C8              LSKP
1056 F801   ONE      LDI    #01
1058 52              STR    2       STORE IN MEMORY FOR 'OR'
1059 0A              LDN    A       GET OLD PIXEL
105A F1              OR               'OR' IN NEW PIXEL
105B 5A              STR    A         AND RESTORE TO VDG MEMORY
105C D5              SEP    5       RETURN


105D                 ORG    1063H
1063 F8E1   SCRNCLR  LDI    #E1     LOAD SCREEN BOTTOM - HI BYTE
1065 BA              PHI    A
1066 F8FF            LDI    #FF       AND LO BYTE
1068 AA              PLO    A
1069 EA              SEX    A       SET X FOR CLEARING
106A F800   CLEAR    LDI    ZEROS   LOAD 'ZEROS' FOR
106C 73              STXD             BLACK BACKGROUND
106D 9A              GHI    A       GET HI ADDR
106E FBD0            XRI    #D0       TO TEST IF DONE
1070 3A6A            BNZ    CLEAR
```

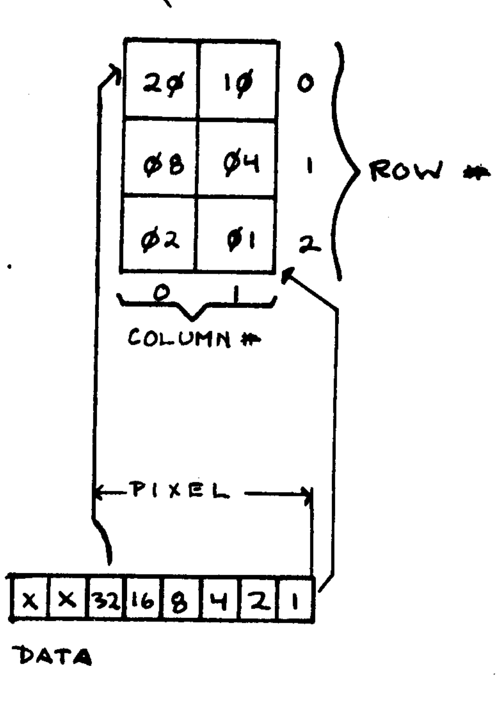
```
1072 E2                 SEX     2               RESTORE X = 2
1073 D5                 SEP     5                AND RETURN

1074                    END
```

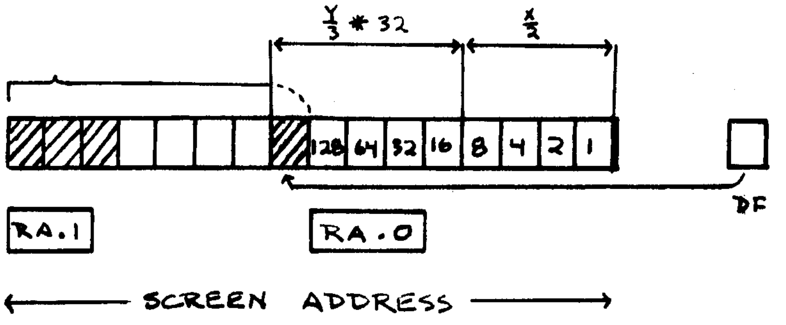
BACKGROUND: FOR POINTSET



FOR ANY X,Y: (3072 PIXELS)

1. LINE # = $\frac{Y}{3}$ QUOTIENT.
2. CHARACTER POSITION = $\frac{X}{2}$ QUOTIENT.
3. ROW # OF GRAPHICS CHARACTER = $\frac{Y}{3}$ REMAINDER.
4. COLUMN # OF CHARACTER = $\frac{X}{2}$ REMAINDER.
5. BIT POSITION = (ROW #) * 2 + COLUMN #





SPRECH - A SIMPLE SOFTWARE VOICE SYNTHESIZER

- by P. G. Liescheski III, 4510 Duval St., #203, Austin, Texas, 78751

SPRECH is an output software package which can give the 1802 a voice. It basically accepts an ASCII numeric character in RF.1 or a binary nibble (least significant) from the accumulator D, and synthesizes the sound of that hexidigit. It is mainly intended as a software novelty; however, it may be quite useful in conjunction with a monitor.

SPRECH is basically a digital voice recorder. Its algorithm is basically similar to that used by Bobby R. Lewis in QUESTDATA (Vol. 2,#2,p. 1). The RECORD routine is used to generate the raw voice data. After manipulation and rearrangement of this data, the TEST and INTERFACE routine can be used to regenerate the sounds of the hexidigits: 0,1,2,3...F. The TALK routine is the basic subroutine which regenerates the sound from the data in memory. TALK performs the inverse function of RECORD.

The most difficult part of this package to implement is the voice digitalization and the voice data manipulation. After this task, SPRECH should be quite simple to use. First, this software must be entered into the 1802 computer. It is assumed that this package will be executed from a monitor which sets R2 as stack pointer, R3 as program counter, and uses R4 and R5 for SCRT Call and Return registers. Some form of audio device such as a tape recorder or an amplifier with microphone must have its output <u>properly</u> connected to the EF3 line. With this the RECORD routine is executed at location 005B. After pushing the I-key, the numbers between 0 and F are quickly but clearly pronounced into the microphone. The recording period should last for about twenty (20) seconds. After this, the memory between addresses 0100 and 4000 is examined. If the amplifier is not too noisy, the memory should be filled with primarily zeros and occasional non-zero patches. These non-zero patches or blocks of memory are merely the digitalized sound of each number. The first block should represent the sound for zero, while the second block should represent the sound of one and so on. The data block for the sound of zero is moved to memory locations 0100-02FF, while the data for one is moved to 0300-04FF, and so on until the number F. The voice data will occupy 8K of memory since the sound for each number can be contained within two pages of memory. After this task, the memory contents between locations 0000 and 20FF should be saved on tape for safe keepings.

Now with this, the voice synthesizer can be used. For hardware, an amplifier with speaker should have its input connected to the Q line (see figure 1). To test the program and data, one should execute the TEST routine at location 0000. With this, one can enter a number on the hexpad. After pressing the I-key, the sound of that number will be regenerated. This routine will allow one to easily check the sound quality of each number.

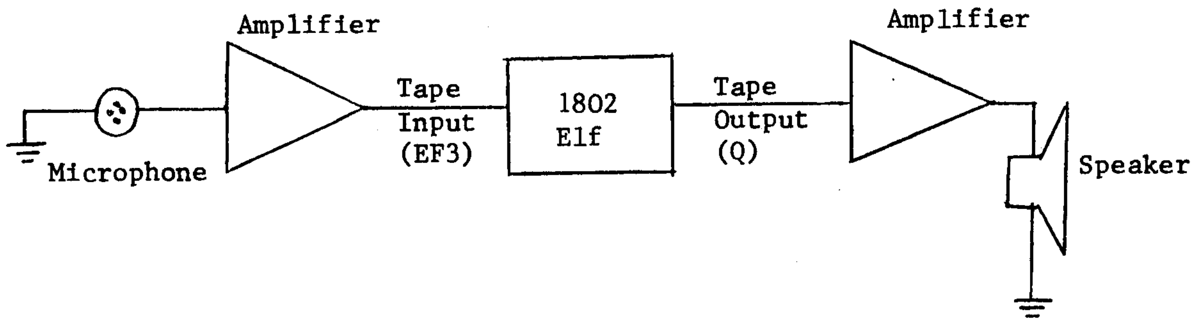
In order to use SPRECH in conjunction with a monitor, one should patch the monitor's output routine so that the INTERFACE routine will be called at location 000A. The routine assumes that the output ASCII byte is contained in RF.1. Also it is assumed that R3 is the program counter; R2 is the stack pointer and SCRT is used. The routine will ignore ASCII characters which are not considered to be numeric.

In its initial testing, the package could reproduce the sound with fair quality. Some problems are encountered with B,C,D and E. These numbers tend to sound the same. This flaw could be the result of a poor audio system. The audio system used in the initial test was a tape recorder connected to the tape I/O ports of the computer. A filtering system as described by James C. Anderson in BYTE (Vol. 6,#2,p. 36) may improve the sound quality. It is hoped that this voice synthesizer package can be put to some practical use.

Register Assignments:

R2	- Stack Pointer
R3	- Program Counter
R4	- SCRT Call register
R5	- SCRT Return register
R8	- 8-bit Counter
R9	- Data Memory pointer
RF.1	- ASCII Output code pass

Basic Hardware Setup:



```
*
* SPRECH * Phillip B. Liescheski III * 10-16-81
*
* TEST - Voice Tester
*
0000: 3F 00       I-key wait delay
  02: 37 02
      6C          Get number from Hexpad (Input 4)
      D4 00 32    Call TALK
      30 00       Do it again
*
* INTERFACE - Monitor Interface
*
000A: 9F          Get ASCII character from RF.1
      FF 30       Check for Non-numeric ASCII code
      3B 25
      FF 0A
      33 19       Check for Numeric ASCII code: 0-9
      FC 0A       Convert ASCII to binary number
  15: D4 00 32    Call TALK
      D5          Return to monitor
  19: FF 07       Continue to check for Non-numeric ASCII code
      3B 25
      FF 06
      33 25       Check for Numeric ASCII code: A-F
      FC 10       Convert ASCII code to a binary number
      30 15       Jump to TALK
  25: F8 FF       Momentary Delay for Non-numeric ASCII code
  27: C4 C4       Delay Loop
      C4 C4
      C4 C4
      FF 01       Bump Delay Counter
      3A 27
      D5          Return to Monitor
*
* TALK - A Routine that regenerates sound from voice data
*
0032: FA 0F      Mask off upper nibble of D
      FE         Calculate page address of voice  data block
      FC 01        which represents the number in D
      B9         Store Page number in R9
      FC 02      Calculate & Store end address on stack
      73
      F8 00      Finish the voice block address in R9
      A9
  3E: F8 08      Set up R8 as 8-bit counter
      A8
      49         Get a byte from voice block using R9 as memory pointer
      52         Store it on stack
  43: 99         Check if finished with voice block
      60
      F3
      C6         Skip return if not finished
      D5         Return
      C4         Fill in the skip gap
  49: 22         Bump stack pointer(assume X=2)
      F0         Get voice byte from stack
      F6         Shift right
      33 4F      Toggle Q according to DF bit
      7A         Q=1 if DF=1
  4F: 3B 52
```

```
       7B          Q=0 if DF=0
  52: 52           Push processed voice byte back onto stack
      28           Bump 8-bit counter
      88           Check if finished with the voice byte
      32 3E        If finished with byte, then fetch the next voice byte from memory
      C4 C4        Keep timing smooth
      30 43        If not, continue as usual with the voice byte
*
* RECORD - A Routine to produce the voice data
*
005B: F8 00        Start recording of voice on page one of memory
      A9           Set up R9 as memory pointer
      F8 01
      B9
  61: 3F 61        Wait for I-key depression
  63: 37 63
  65: F8 08 A8     Set up R8 as 8-bit counter
      19           Bump memory pointer R9
  69: 99           Check if finished with recording
      FB 40        Last recording page of memory is 40
      C6           Skip halt if not finished
      00 00        Halt!
  6F: 49           Get byte from memory
      F6           Shift right
  71: 3E 75        Check the EF3 line
      F9 80        If EF3=1, then set most significant bit of D
  75: 36 79
      F9 00        If EF3=0, then reset most significant bit of D
  79: 29           Bump memory pointer
      59           Store byte back in memory for safe keepings
      28           Bump 8-bit counter
      88           Check if finished with this byte
      32 65        If so, start working on a new byte
      88 A8        Keep timing smooth
      30 69        If not, continue as usual
*
*
*
```