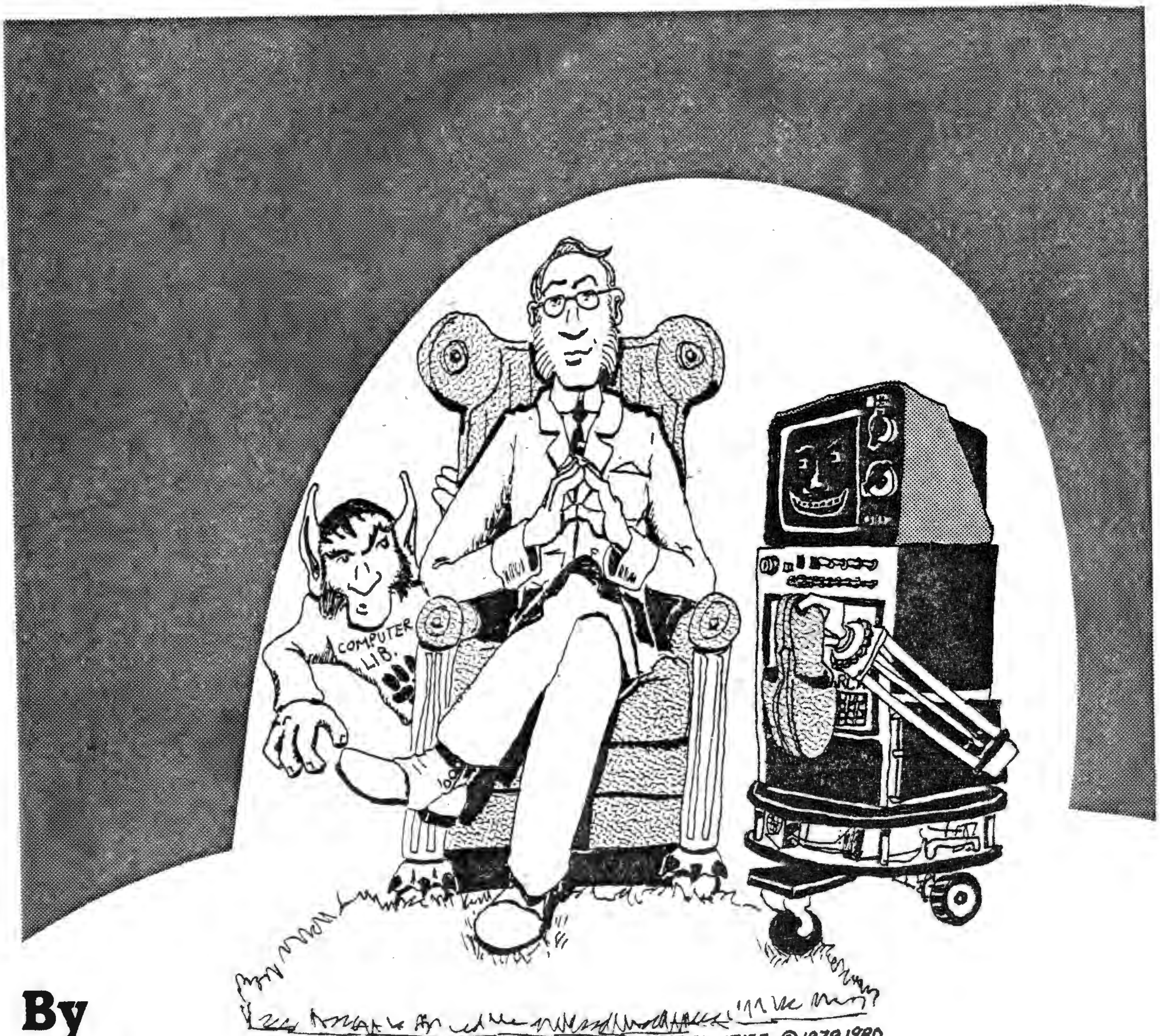
CAPTAIN COSMO'S WHIZBANG



COVER DRAWN BY CHRIS CLOUTIER, ©1979,1980

By Jeff Duntemann

For Me and You and the 1802 !

WHAT IS THIS? It's a book, by cracky, about the 1802; hopefully the oddest and most entertaining book on any microprocessor ever written. The 1802 is, after all, an odd and entertaining chip. This view is not shared by all. Physicist Mike Brandl said he could swallow a mouthful of sand and barf up a better microprocessor than the 1802, and another colleague claims its instruction set demands that he program with his left hand. Bitch, bitch, bitch. I kinda like it. Much of this material I doped out while recovering from hernia surgery not long ago and couldn't lift anything heavier than a 40-pin DIP. I had a lot of fun and thought you might like to be copied in on it.

Like everything else I do, this book is an experiment. If I don't take a serious loss on production and mailing costs, I may do up another one. I've got a little gimcrack on the bench that'll make you people drool: an easy-to-build thermal printer for the 1802 that you can make for seventy bucks flat with all new parts. I'm working on an automatic phone dialer board and a few other things. Selectric interface. Robotics. Ham radio stuff. All kindsa things. Are you interested? Would you lay out another five beans for a Volume II? Let me know; drop me a note with any and all comments and spare not the spleen; I'm a hard man to offend and I love crackpot letters.

I might also consider buying small programs for publication if I find a market for this thing. Cartoons, too; I love cartoons; no book or magazine worth a bran muffin lacks cartoons. Whatever you do, send me copies if you're sending me things; never send originals. I can't promise to send them back. I will promise not to print anything of yours without making arrangements with you. I may be weird, but I'm honest.

ABOUT THE AUTHOR: Jeff Duntemann still has his appendix, tonsils, and all of his teeth except one which never grew in. He wears a size 8 medium bowling shoe and hates the living hell out of Disco. He messes with anything electronic, is nice to women, and goes berserk over a good pepperoni pizza. If you know what FIAWOL means you'll know where to find him every Labor Day and at other times during the year, and no, if you don't know he's not going to tell you.

(There are at least 2 1802's on Mars! Eat mundane death, Z80!!)

Here a flip;
There a flop;
Everywhere a no-op...

It was a couple of years ago that I was riding the Clark St. bus with Gus, a friend of mine. He was reading the new Popular Electronics over my shoulder. We were looking at a picture of a piece of perfboard with a bunch of IC's on it, held in a man's hands. It was supposed to be a computer. Gus saw my interest and shook his head. "No good," he said. "It multiplexes the address bus."

"Oh," I said, and turned the page. I knew of computers only as complicated digital devices that moved bits around in memory, added a lot, and generally ran things. I had no idea how they worked. I had read snippets in various magazines about vectored interrupts, real-time clocks, bidirectional data busses, and so on. Nobody said much about what a computer actually *did.* And I'll never forget the incandescent blaze of understanding that roared out of my ears when another computer freak friend finally explained what an instruction set was, and what a computer did with a series of instructions stored in memory.

EPIPHANY!!!! A COMPUTER IS A BOX WHICH FOLLOWS A PLAN! (to steal a phrase from Ted Nelson.) With that out of the way, I knew I had to have one. Quick flip back to the man holding the perfboard computer. COSMAC ELF. Screw the multiplexed data bus. I was starting from Square One, and what I didn't know wouldn't hurt me. The damned thing would cost me less than sixty bucks. Now *that* I understood.

And if it didn't work, I could always make an ashtray out of it.

HAMFESTS--Now, you may already have your ELF built. But if you haven't built it yet, or are still gathering parts, give a listen. My ELF cost me fifty bucks total. I saved a bundle just by haunting hamfests. A hamfest is a flea market devoted almost entirely to electronics junk. It's put on by radio amateurs, or "ham radio ops" as we are called. We're the guys who plant aluminum jungles atop our houses and screw up your idiot box at all hours of the night. I'm KB2JN . When are you going to get *your* license? Anyway, at hamfests guys set up tables piled high with the scrapings off their workshop floors, Ur Choice 25 cents. Surplus dealers hold special sales, and the *damndest* things can be had, usually for a song. I recall going to a hamfest soon after deciding to build my ELF. I saw a boxfull of LED readouts that said HEXADECIMAL over them. No price. TIL-311's. Hmmmmm. "How much?" I asked the kid who was manning the tables, maybe twelve years old and looking bewildered. "Oh...uhhh...dollar on all the digit readouts." I shrugged and handed him a tenspot. "How 'bout twelve for ten?" I asked. He rolled his eyes, obviously thinking how pleased his daddy would be to see such salesmanship. "Sure." And that's how I took home twelve TIL311's for 83 cents each. Later on I bought a wire-wrap board full of sockets--and full of wire. For several days I commuted to work on the Howard St. El with the damned thing on my lap, ripping and tearing at the jungle of old wire with a borrowed unwrapping tool. Cost me $3.50--and on it I built a helluva computer. The winos on the subway thought I was crazy.

So check out the hamfests. How to find them? Read the ham magazines: *QST*, *73*, *CQ*. These are often sold at computer stores and electronics stores. Each has a listing of hamfests for the coming months. Make up a shopping list, and go to it. You may occasionally get burned by unscrupulous dealers. I bought six LM380's for a dollar once--and got six burned out LM380's. All I know about the dealer is that he was an ugly asshole. May the Zap Fairy fill his bed with stonkered IC's, pin side up. Beware. But by and large, hamfest material is all right.

IT MAY OR MAY NOT COME IN THE MAIL--I'm constantly in touch with a large number of basement tinkerers who buy large quantities of suplus electronics. What I'll do here is share some of their and my experiences with mail-order dealers. If a dealer isn't meantioned here, that only means No Data Available. Give 'em a try. It's a competitive field, and jerky dealers don't last long.

The finest of the surplus dealers is unquestionably Digi-Key. Low prices and superb service, quick shipping, and fine material. I have *never* gotten a faulty item from them in over 25 orders. Their handling fee is only 75 cents, and they back-order only rarely. I buy all my CMOS (except COSMAC--they don't carry it) from them. James is also good, and very fast shipping. Their prices are higher than they should be, however. I suggest buying COSMAC items from Quest or Anacrona. Quest has a good record, and although their shipping is a little slow, their prices are good and I have never received a bad item. Advanced Computer Products has a tremendous selection of IC's and computer paraphenalia. Their catalog is a must-have. One of my friends got screwed over by them, but they have always given me good service. Tri-Tek, home of Ampl'Anny, is excellent but has a rather limited selection. Godbout has superior service but high prices. Jade is erratic. Two of my friends have gotten faulty IC's, but others have been satisfied. Integrated Circuits Unlimited has little computer stock, but their prices are low, service fine, and have no minimum order. Poly Paks is a real screwball outfit. Reading their catalog takes some experience. When they say something has "100's of uses" it's probably worthless. "100% hobby" means 100% shit. Their LED products are dismal. They frequently describe various parts as "great for projects;" I suppose a lot of other people sprinkle them on their corn flakes. Their IC's are generally OK but overpriced. Resistors and caps, OK. The catalog is worth having because they do sell a lot of screwball stuff unavailable elsewhere, but if they send you crap--send it back, and demand a refund. You'll get it.

UNDER NO CIRCUMSTANCES buy from Ace, Edlie's, TK Graphics, MiniMicroMart, Delta, ETCO, Formula International, or Trinico. These companies have screwed over people I have known, and I'm trying to drive them into bankruptcy. Do your part. Send your bucks elsewhere.

CMOS RAM BARGAIN--New Tone Electronics sells prime NEC 5101-3 CMOS RAM for $4.50. This is the RAM buy of the decade. Real CMOS that you can keep alive with a couple of hearing-aid batteries almost forever. These are plug-compatible with 2101's--you can use them interchageably. Unless you're starving on welfare, spend the extra couple bucks and have an all-CMOS system. You won't regret it. With the TIL-311's pulled, my ELF ran at 3.579 Mhz and drew only 25 milliamps. Try and beat that!

WIRING WOES--There is only one way for a basement tinkerer to build a computer. By wire-wrapping. Period. (Unless you buy a kit or readymade PC, but that isn't *real* tinkering...) Trying to design your own PC board is silly. You can't re-fry an egg, and you can't re-etch a PC board. Make one little trace wrong, and...you have to bridge gaps and fix booboos with little wire-bridges. Do enough of that, and the thing is going to start looking wire-wrapped anyway. Do it right. Get a hobby-wrap *modified* wrap tool, some wire-wrap wire, and start wrapping. Practice a little first; there *is* a trick to it. But the magic of it is that you can take a whole subsystem of the computer apart and redo it without scrapping the whole board. You can add features to (or dump them from) your computer at any time. You can experiment, try this and that, and be a *real* hardware freak without making a whole new computer for each modification. People sometimes object to wrapping because it's not a firm, unshakable solder connection. BS. The torque on those battery-operated guns is so high, that the resulting wrap is damned near airtight. The wraps turn black after awhile, because wire-wrap wire is generally silver-plated. Silver tarnishes. Big deal. Silver tarnish conducts electricity! (Failed Chem 101, huh? It shows.) Maybe it looks flimsy, and somehow the chaos of a densely wrapped board offends your prudish orderly sensibilities. Tough luck. It works. You're a damned fool if you do it any other way.

WATCH OUT--A common mistake to new computer builders lies in toggle-switches. When I built my ELF, I wired my toggles upside-down, thinking that when the toggle handle is up, that the center terminal is connected to the top terminal. Wrong-o. Think of it like this: The toggle handle points to the two terminals which are shorted together. Like in the diagram below. This problem was compounded for me by the fact that I drove my TIL311 hex display chips with inverting CMOS drivers, CD4049's. (Sheer stupidity on my part.) The toggles were inverting the binary bits they were feeding the computer, and the 4049's were re-inverting those bits for my eyes, so that while everything looked normal, nothing ran right. Hence I contracted my first case of The-Chip-Is-Bad Fever, when nothing can possibly be wrong except for a bad CPU. Don't you believe it. More on TCIB Fever later on.

GIVE YOURSELF ROOM--If you haven't already started your ELF, fer chrissake get a bigger piece of perfboard. There's nothing sacred about the parts layout shown in the PE series, except possibly to keep the crystal close to pins 1 and 39. The other stuff you can put anywhere it fits, especially if you wire-wrap. 5 1/2 X 8 is a good size. Don't let all the empty space bother you. There's plenty of things you can add on later to fill things up.

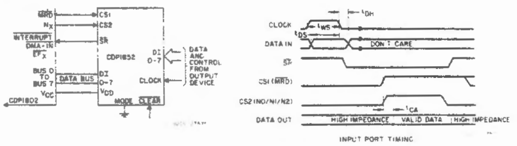
RAM CHOICES--Whatever you do, don't use the RAM expansion circuit given in PE March '77. It's hopelessly awkward and needlessly time-consuming. Remember, that was 3 years ago. Today we have the 2114 1024 X 4 static RAM for eight bucks. Two of them will give you 1K RAM if you include a CD4042 latch to save address lines 8 & 9. If you haven't started your ELF, use the circuit in Fig. 2 instead of 2101 RAM. If yours is built, I think it would be worth ripping the 2101 sockets out and replacing them with a pair of 2114's. (Which are smaller, physically, than 2101's.) Now you see the merits of wire-wrapping!

POWER SUPPLY POINTERS--It's silly to scrimp on power supply capacity. An LM340K-5 costs you a buck at most, and will supply an amp and a half if properly heat-sunk. Use the one in the big power-transistor TO-3 case. It will permit your ELF to support a lot of accessories and peripherals, like an ASCII keyboard and paper tape reader, as well as lots more RAM. Another idea is to use a 6 volt motorcycle battery instead of an AC supply, and just recharge the battery every few weeks. I have done this to no ill effect on my ELF, even with TTL chips on the board. Also, your computer will then be portable. Just don't get battery acid in the wiring. Four AA nicads make a dandy ELF supply. If you plan to add a lot of accessories, use four C nicads. This will give you a computer you can throw in your briefcase to while away the hours on long bus, train, and car rides. (Using it on a plane might be frowned upon. Especially if the captain hears your switching transients on his radiolocation unit.)

COSMAC I/O PORTS--I suspect very few people have used the CDP1852 COSMAC I/O port because of its steep $9.50 price. It's a good unit, and now available from Anacrona for $1.85 in the new 'consumer plastic' version. (more on that further down) It performs the function of either of the CD4508 ports shown on page 38 of PE 9/76, with the advantage that all those NAND gates are included on the chip. Also, there is a CLEAR input which will set latch contents to zero. When CLEAR goes low, the data registers are cleared.

The diagram should tell the whole story. You can use either the input port or output port separately; they don't depend on one another for operation.

8-Bit Input/Output Port

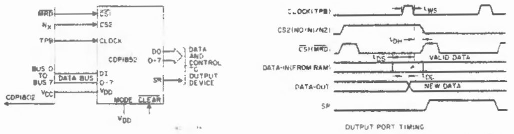


MODE = 0

CLOCK	CS1·CS2	$\overline{\text{CLEAR}}$	Data Out Equals
X	0	X	High-Im-pedance
0	1	0	0
0	1	1	Data Latch
1	1	X	Data In

$\overline{\text{SR}}$ = 0	$\overline{\text{SR}}$ = 1▲
CLOCK	CS1·CS2 Or $\overline{\text{CLEAR}}$

Fig. 2–CDP1852 input port operation.

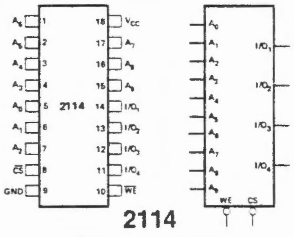


MODE = 1

CLOCK	$\overline{\text{CS1}}$·CS2	$\overline{\text{CLEAR}}$	Data Out Equals
0	X	0	0
0	X	1	Data Latch
X	0	1	Data Latch
1	1	X	Data In

SR = 1▲	SR = 0
$\overline{\text{CS1}}$·CS2	CLOCK Or $\overline{\text{CLEAR}}$

Fig. 3–CDP1852 output port operation.

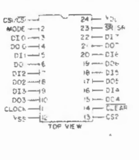


2114 STATIC RAM

PIN NAMES

A_0–A_9	ADDRESS INPUTS	V_{CC} POWER (+5V)
$\overline{\text{WE}}$	WRITE ENABLE	GND GROUND
$\overline{\text{CS}}$	CHIP SELECT	
I/O_1–I/O_4	DATA INPUT/OUTPUT	

CDP1852CD



Terminal Assignment

1K - The easy way!

GROUND ALL UNUSED CMOS INPUTS ON 4042, 4081

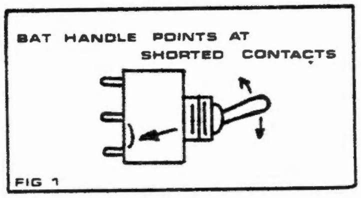


FIG 1

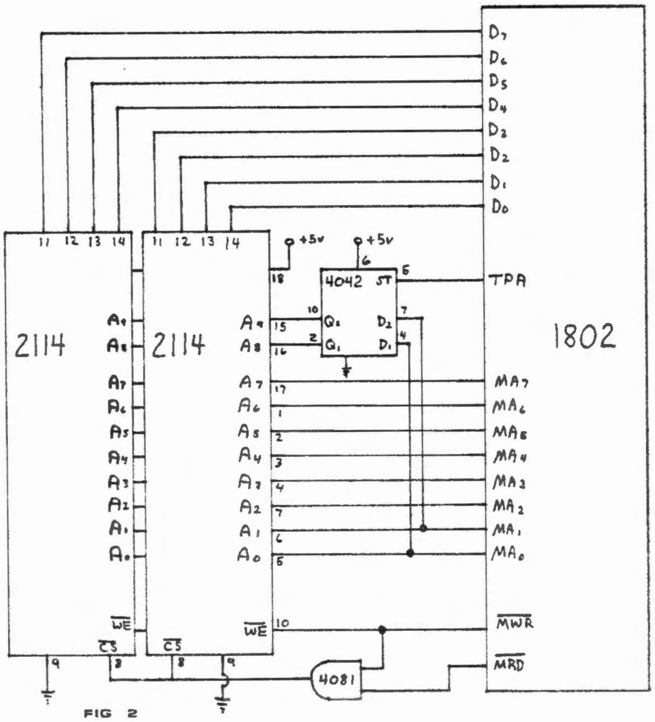
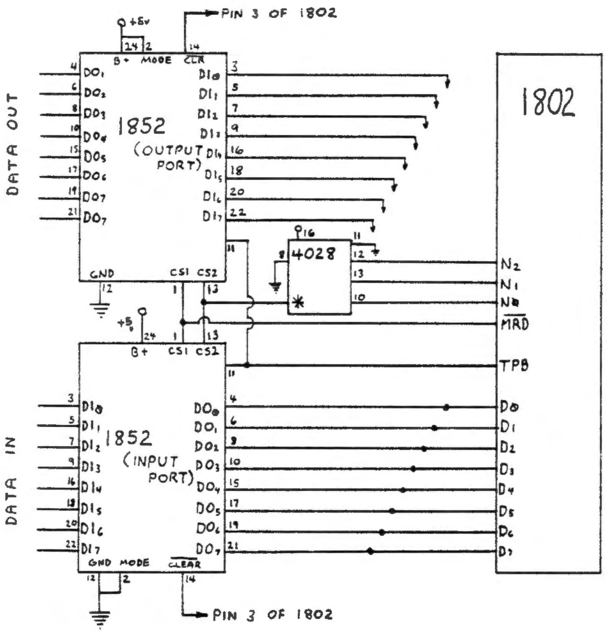


FIG 2

If you are very sure you will not be needing more than 3 input ports and 3 output ports, you can do away with the 4028 and simply connect one of the N lines to the two CS2 pins. Since a 50 cent IC more than doubles your I/O capacity to 7 of each kind of port, you're silly not to use the 4028. The asterisk'ed connection is up to you: you choose which pair of I/O opcodes you want to control the ports, look up the binary code presented by the N lines for that pair of opcodes, and then connect the CS2's to the 4028 output line which decodes that binary code. That same 4028 can handle all your other I/O decoding as well; it is not limited to the pair of ports shown here. Remember to ground pin 11 of the 4028!

1852 I/O System



*= SEE TEXT

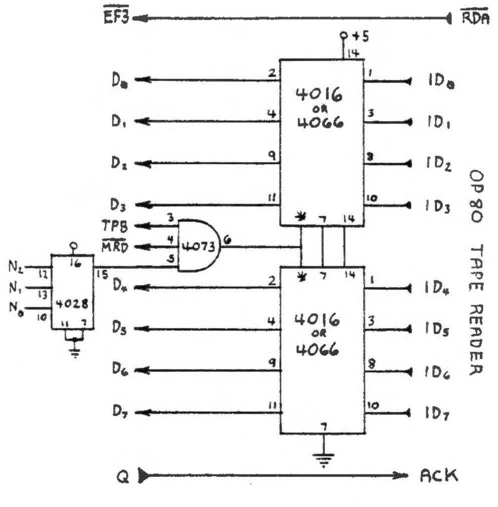
EVERYTHING YOU NEVER WANTED TO KNOW ABOUT PAPER TAPE-- Lack of a good casette recorder to experiment with led me to a financially disastrous love affair with paper tape. First I bought an OAE hand-powered paper tape reader and punched tapes on a borrowed teletype. This worked so well I bought the incredibly, impossibly awful Heathkit H10 paper tape reader/punch. Why? Simple. I had to give the teletype back, and I needed tapes. The problem was, theHeath punch could not space its rows of holes consistantly. Consequently, it could not read its own tapes. No matter how much I jiggered it (and I am a damned good jiggerer) the punch would not come across. Heath claimed I assembled it wrong. This may be true. I doubt it. Caveat Aemptor.

In any case, the following pages contain some software for use with the OAE paper tape reader, which I must admit is a good piece of gear. The general system for using the OAE reader is this: The first sixteen bytes of ELF memory is reserved for the bootstrap loader. You toggle this in by hand. Whatever you load with the boot must be written to run at M(00 10). Generally this is a simple op system, to enable you to read/write and tape load into any other memory location. I have modified EHOPS and ETOPS to include paper tape load abilities, plus high-order register housekeeping so they will run on a system with more than 256 bytes of memory. If you have a teletype with a tape punch, the OAE reader is not bad, but with cassette storage as cheap and easy as it has become, paper tape has a touch of the dinosaur in it. I don't think I'll ever go back to it.

OP-80 Bootstrap Loader

```
MA  OPCODE

00  90 BF F8 10 AF  Set RF to 0010
05  3F 10           Jump to pgm unless INP
07  3E 07           Wait for RDA
09  EF              Point X to RF
0A  6B              Input frm reader to M (RF)
0B  64              Display byte & inc. RF
0C  7B 7A           Generate reader reset pulse
0E  30 07           Wait for next RDA
```

Interface for the OP-80A Paper Tape Reader



* PINS 5,6,12,&13 ARE TIED TOGETHER FROM BOTH CHIPS

VERSION, VOLTAGE, & SPEED--There is no difference whatsoever in the way RCA manufactures the CDP1802CD and the CDP1802D. It's the same chip. Individual differences in overall chip leakage determine how high a voltage each chip will run at without drawing excessive current and overheating. You have to remember that voltage and voltage alone determines how fast a CMOS chip will run. To make an 1802 run at five megs, you have to crank the voltage up to ten volts. If you crank a 'CD' version chip up to ten volts, it will probably fry in its own leakage dissipation. Ergo, it takes a 'D' version to run at five megs, only because its leakage is low enough to handle a ten volt supply. RCA tests each 1802 as it comes off the line with an ammeter in series with the supply. As the supply goes up, the current draw' will go up, and somewhere along the way each chip 'redlines.' The chips that redline around seven volts are stamped 'CD'; those that make it all the way to fifteen volts become 'D's. Like Gus Flassig used to say, everybody makes 5% resistors; those that don't make the grade get silver bands or no bands at all.

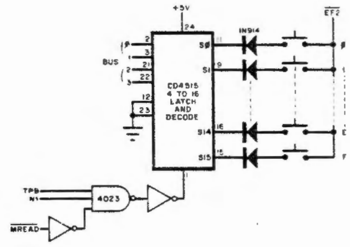
I say this because I've heard of people paying for the premium 'D' chips and thinking they can put a six meg crystal in their ELF and run super speed at 5 volts. Won't work. Assuming a 5V supply, when your clock speed goes much above 2 megs, two things begin to happen: First, the internal crystal oscillator may not start, and once started, may run erratically. You can check this yourself with a good scope; up past 2 megs your square wave is anything but square. Secondly (and more seriously), at higher speeds your TPA pulse starts running late. Eventually it will occur during the time the address lines switch from high address to low address, and your address decoding system will start decoding indeterminate addresses. In other words, your computer goes berserk. This will happen no matter how good a chip you have. To run fast, you have to crank up the supply voltage. There's just no way around it. So if you do up the supply, make damned sure your other chips can handle it! (TIL311's ain't cheap!!!!)

TEHOPS 65K   TAPE ENTERED HEXADECIMAL OP SYSTEM

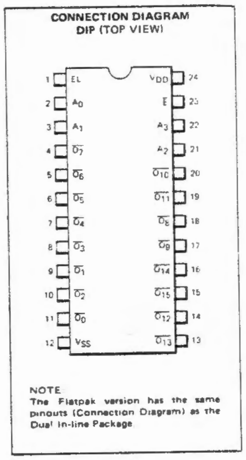
by Jeff Duntemann   2/3/78

```
MA  OPCODE

10  F8 00 B2 B3 B5 B6      Initialize high order registers
16  F8 FF A2               Initialize stack pointer
19  F8 23 A3               Initialize MAIN PC
1C  F8 4E A5               Initialize BSUB PC
1F  F8 5E A6               Initialize HSUB PC
22  D3                     P=3
23  D5 B1 D5 A1            Enter starting address from keyboard to R1
27  6C                     Input toggles to stack
28  3A 33                  If toggles are not 00, go to M(33)
2A  F8 2E A0               Repoint R0
2D  D0                     P=0
2E  91 B3 81 A3            Transfer R1 into R3
32  D3                     P=3 (jump to execute program at M(R(3)))
33  F6                     Shift D one bit right
34  3B 3B                  If DF=0, go to M(3B)
36  D5                     Call BSUB
37  E1                     X=1
38  64                     Display M(R1)) and increment R1
39  30 36                  Loop again and display next byte
3B  F6                     Shift D one bit right
3C  3B 44                  If DF=0, go to M(44)
3E  D5                     Call BSUB
3F  E1                     X=1
40  51                     Store D at M(R(1))
41  64                     Display M(R(1)) and increment R1
42  30 3E                  Loop again_and store next byte
44  3E 44                  Wait for RDA
46  E1                     X=1
47  6B                     Input byte from tape reader to M(R(1))
48  64                     Display inputted byte and increment R1
49  7B 7A                  Generate reader reset pulse
4B  30 44                  Loop again and input next taped byte
                      BSUB
4D  D3                     Return from BSUB
4E  D6                     Call HSUB
4F  FE FE FE FE            Shift D four bits left
53  A0                     Store D in R0.0
54  D6                     Call HSUB
55  80                     Fetch contents of R0.0
56  F1                     M(R(2)) OR D, put in D
57  52                     Store D at M(R(2))
58  64 22                  Display stack and decrement stack pointer
5A  30 4D                  Go to Return
                      HSUB
5C  F0 D5                  Return from HSUB
5E  E2                     X=2
5F  FC 01                  Add D+1, put in D
61  FA 0F                  D AND 0F, put in D
63  52                     Store D at M(R(2))
64  62 22                  Output byte to CD4515 scanner chip; dec. R2
66  3D 5E                  Loop again if no key found pressed
68  7B                     Turn Q on
69  F8 09 B4               Load debounce timing constant
6C  24 94 3A 6C            Decrement constant and test for done
70  7A                     Turn Q off
71  35 71                  Wait for key released
73  30 5C                  Go to return
```

HEX KEYBOARD CIRCUIT



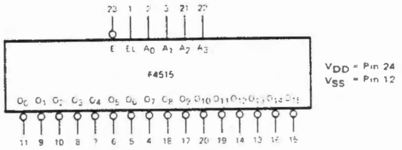
4515



CONNECTION DIAGRAM
DIP (TOP VIEW)

NOTE
The Flatpak version has the same pinouts (Connection Diagram) as the Dual In-line Package

LOGIC SYMBOL



V_{DD} = Pin 24
V_{SS} = Pin 12

HOW TO USE THIS PROGRAM:

This op system is actually an adaptation of the EHOPS-256 op system given in the original ELF series in PE. It requires the scanning hex keyboard described in the second article, without any modification. It also requires an input port strobed by the 6B INPUT instruction.

Operation is very similar to that of EHOPS-256, with the important exception that a two byte starting address must be entered from the hex keyboard. The toggle commands for write, inspect, and run are identical to EHOPS-256: 02 for write, 01 for inspect, and 00 for run.

04 is an additional command which will cause the program to input a string of data from an OP-80 paper tape reader to any location in memory with the first byte input to the starting address. To input from tape, position the tape on the reader so that the first byte you wish to input is immediately to the left of the phototransistor strip. Make sure the sprocket hole sensor is covered by tape and that the SP lamp is off. Set 04 into the toggles and flip RUN up. Enter the two-byte starting address with the higher-order byte first. Then pull the tape through the reader, taking care to keep the tape flat over the phototransistor strip. As the tape moves through the reader, the data being input will appear on the hex displays.

If you input a program from tape and it will not run properly, load it again a little more slowly before assuming something is wrong with either program or hardware. If you still have no success, inspect memory at the starting address to see if the first byte of the program was in fact inputted. If you positioned the paper tape with the first byte directly over the phototransistor strip, the computer will skip the first byte and begin by loading the second byte in the starting address. Move the tape a bit to the left and load again.

Remember that any program run through this op system will use R3 as its main program counter. Also, R5 and R6 cannot be used in your programs if at any point in the program you intend to call BSUB. However, BSUB may be called any time hex input is desired if those registers are respected.

TETOPS 65K   TAPE ENTERED TOGGLE OP SYSTEM

by Jeff Duntemann   2/10/78

```
MA  OPCODE

10  F8 00 B2 F8 49 A2    Initialize stack pointer R2
16  E2                   X=2
17  6C                   Read & store toggles on stack
18  64                   Display toggles & increment stack pointer
19  3F 19                Wait for INPUT pressed
1B  37 1B                Wait for INPUT released
1D  6C                   Read toggles
1E  31 24                Go to M(24) if Q is on
20  7B                   Turn Q on
21  B3                   Put D in R3.1
22  30 19                Go to M(19)
24  A3                   Put D in R3.0
25  7A                   Turn Q off
26  22 F0                Fetch function byte from stack
28  3A 2B                Go to M(2B) if D≠0
2A  D3                   Jump to execute at M(R(3))
2B  F6 33 3B             Test first function bit; branch if 1
2E  F6 33 3A             Test second function bit; branch if 1
31  E3                   X=3
32  3E 32                Wait for RDA signal from tape reader
34  6B                   Read & store tape at M(R(3))
35  64                   Display tape & increment R3
36  7B 7A                Generate reset pulse for tape reader
38  30 32                Go to M(32)
3A  7B                   Turn Q on
3B  E3                   X=3
3C  3F 3C                Wait for INPUT pressed
3E  37 3E                Wait for INPUT released
40  31 45                Go to M(45) if Q is on
42  64                   Display M(R(3)); inc. R3
43  30 3C                Go to M(3C)
45  6C                   Read & store toggles at M(R(3))
46  64                   Display toggles & increment R3
47  30 3A                Go to M(3A)
```

HOW TO USE THIS PROGRAM:

This is an operating system which will handle up to 65K of RAM. It requires an OP-80 paper tape reader interfaced through an input port strobed on the 6B INP instruction. To write in memory, set 02 in the toggle switches before running the program. Flip RUN up. Enter the higher-order byte of the address at which you wish to begin writing. Push INPUT. Q will come on. Enter the lower-order byte of the address, and push INPUT again. Your address is set. Begin writing at that location by entering a byte in the toggles and pushing INPUT. Each byte will be stored at the next highest memory location.

To sequentially inspect memory, set 01 in the toggle switches. Flip RUN up. Enter the starting address as above. Once your starting address has been set, push INPUT. The contents of memory at your starting address will be displayed. Pushing INPUT will display successive memory locations. The byte set into the toggles while displaying is not important.

To input a program or data string from paper tape through the OP-80 reader, position the paper tape on the reader so that the first punched byte you wish to input is immediately to the left of the reader's phototransistor strip. Be sure that the sensor spot which reads the sprocket holes is covered by paper between holes, and that the SP lamp is off. (Be sure, though, that the lamp lights when the sensor spot is exposed to light.) Once the tape is in position, set 04 into the toggles, and flip RUN up. Enter the starting address of the program or data string as above. Then pull the tape through the reader, taking care to keep the tape absolutely flat over the phototransistor strip. When you see that the punched portion of the tape has been pulled completely past the phototransistor strip, stop pulling and halt the computer. Pulling past the end of the significant data will load 00's in memory, possibly over other programs or data.

To run a program which has already been stored in memory, set 00 into the toggles and flip RUN up. Input the starting address as above. The program will begin running immediately after entry of the lower-order address byte. R3 will be the program counter, regardless of how the program was written. Be careful if you use subroutines that your routines return to MAIN program counter R3. Returning to other registers as MAIN program counter may lead to halting, endless loops, tapeworms, and other bizarre computer behaviour.

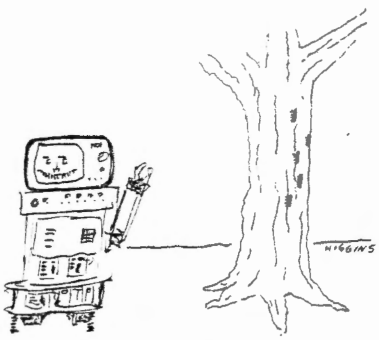
If you load a program from paper tape and it will not run properly, load it again, perhaps a little more slowly, before assuming something is wrong. Then, if you have no success, check to see (by inspecting memory) whether the first tape byte was actually loaded. If you positioned the tape so that the SP lamp was on when you hit RUN, the program will skip the first tape byte. Move the tape a hair to the left and load again.

THE 1802 MASS STORAGE PROBLEM--I don't really recommend paper tape, but if you have a Teletype with a punch, what the hell. Unless you stomp on it, the stuff is pretty well indestructable, and in skilled hands reads as reliably as cassette. I have had difficulties with cassette machines that make me distrust them as well, but they're pretty much what we have. You people who also have a "big" machine with floppy disk capability ought to consider some sort of program for storing 1802 programs on disk, and reading them into the 1802 through a parallel port. I am currently working out the means to feed an eighty-dollar ELF from a $4000 S100 Z80 mainframe system. Is this silly? Well, that depends. Both are computers, regardless of cost. And while I can't expect my ELF to compute Fourier transforms, I also can't expect my mainframe to hide inside Cosmo the robot. Viva la difference!

VANISHING BOOTSTRAP LOADER FOR OP-80 PAPER TAPE READER

by Jeff Duntemann   2/21/78

```
MA  OPCODE

00  90 A4             Point R4 at 00
02  F8 FF AF          Point RF at FF
05  F8 19 A3          Point R3 at 19
08  EF                X=F
09  03                Load M(R(3)) into D
0A  32 F7             If D=00, go to M(F7)
0C  73                Store D at M(R(F)) & decrement RF
0D  23                Decrement R3
0E  30 09             Go to M(09)
10  00                Null
11  E4                X=4      ___
12  3E F8             Wait for RDA
14  6B                Input byte from tape reader to M(R(4))
15  64                Display inputted byte; increment R4
16  7B 7A             Create reader reset pulse ___
18  30 F8             Go back and wait for next RDA
```



Cosmo's Tips:
MOS can often be found
on the north side of trees.

HOW TO USE THIS PROGRAM:

This program was written for single-page memory systems only. Its purpose is to load paper tape software which is located at M(00) and up. The program contains a bootstrap loader, and, when executed, writes this bootstrap loader into the very top end of the memory page. As soon as the loader has been relocated, it is executed, and waits for the first RDA signal from the reader. Load this program through the toggles, and before running it, position the paper tape on the reader. After flipping RUN up, pull the tape through the reader. The reader will load the taped software starting at M(00).

If you wish to enter another program via the bootstrap, you must toggle in 90 A4 30 F7 before running again, to repoint the bootstrap data pointer and jump to the bootstrap itself.

HOW MUCH RAM?--A lot of hardware oriented people go ape-shit building their first computer, particularly a computer like the ELF which has no assembler available for it. They go to great lengths adding 4K RAM and then spend all their time diddling with shortie programs that never make it out of the first memory page. Be realistic. If this is your first computer, you damned well won't need more than 1K for quite awhile. With 2114's as cheap now as 2101's were in 1976, you may as well start right and build your ELF using the circuit below. It requires no complex memory-decoding systems, and will give you more than anough room to get hopelessly lost in.

HOW FAST RAM?--Another good question. An 1802 newcomer generally notices that COSMAC's clock runs fast. Up to 6.4 Mhz, in fact, using the D version at 10V. The machine should really rip, huh? Well, there's a joker in it. Each COSMAC machine cycle takes eight clock cycles. And each instruction takes at least two machine cycles. The Z80, on the other hand, needs no more than 5 cycles to execute an instruction, with a couple of exceptions. Lay that against the 1802's minimum 16, and you'll see that the 1802 is way behind in the uP speed sweepstakes. Maximum allowable 1802 memory access time is 4.5 clock cycles. The table below translates this figure into a maximum memory access in nanoseconds for various clock speeds. Surprising, huh? It can work to your favor, though, because slow memory is often very cheap.

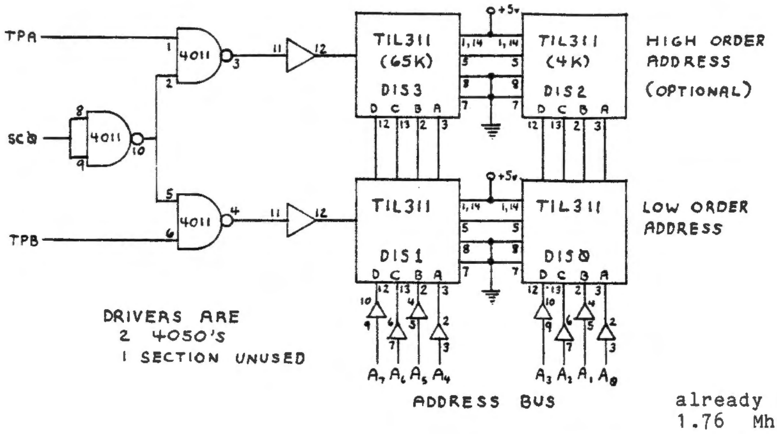
Clock Speed	1 Clock	4.5 Clock	Use At Least
875 Khz	1.14 uS	5.13 uS	2.5 uS
1 Mhz	1.0 uS	4.5 uS	2.0 uS
1.5 Mhz	.67 uS	3.0 uS	1.5 uS
1.76064 Mhz	.57 uS	2.56 uS	1.0 uS
2.0 Mhz	.5 uS	2.25 uS	1.0 uS
2.5 Mhz	.4 uS	1.8 uS	850 nS
3.579 Mhz	.28 uS	1.26 uS	650 nS
4.0 Mhz	.25 uS	1.125 uS	650 nS
5.0 Mhz	.20 uS	.9 uS	450 nS
6.0 Mhz	.17 uS	.75 uS	350 nS

One thing you do have to remember in figuring memory access time is to add in the propagation delays through any latches or gates which lie between the address bus and the memory address inputs. Time through a 4042 latch is about 150 ns max, with the newer latches quite a bit faster than those manufactured more than five years ago. Propagation delay through simple NOR/NAND/gates is almost always less than 40 ns and can usually be disregarded. Access time on most static RAM these days is less than 450 ns; damned few take longer than 800 ns. This gives you a lot of elbow room, so unless you're going to a complex, large RAM system with a lot of decoding, buy the slowest, cheapest RAM you can find and don't worry about it.

WHERZAT PROGRAM AT?--Let's say you've carefully toggled in that 256 byte wonder you stayed up all last night writing, flip the infamous RUN switch up, and then--- What? Nothing happens. What is that beast doing? That's pretty simple. It either stopped dead in its tracks somewhere (usually by hitting a 00 byte) or it's chasing its tail in a blind loop with no exit. Swell. But how do you know which, and where? Easy. Build a memory address display. The circuit is given below.

This circuit displays what instruction the computer is fetching at any particular time. If you run a program and the displays read 00 2F with no ambiguity about any of the numbers, then your program has screetched to a halt having executed the instruction at 00 2F last. What's more likely is that the numbers on the display will look funny. This means that the computer is looping, and it is displaying a series of addresses (addresses of all instructions within the loop) so quickly that your eyes are laying the numbers on top of one another. The way out of this one is to flip run up and down quickly, and note down the resultant address display. When you flip RUN down you stop a program in its tracks, with the display giving you the instruction it was fetching when you hit the switch off. If you do this often enough, the laws of chance say you will eventually see every address within the errant loop. If one or more addresses seem to come up a great deal more frequently than the others, you may well have a loop within a loop. Keeping track of these stopping addresses will eventually tell you exactly where the computer is wandering. How you

ELF Address Display



fix it is your problem. But an address display is one of the simplest and most powerful debugging aids you can add to an ELF. The hex displays are expensive as hell, sure (unless you went to the same hamfest I did) but you will get your money's worth.

Note that in a minimal 256 byte ELF you only need the first two displays, DIS0 and DIS1. Three displays will take you up to 4K, and four will give you all 65K of memory space. So take heart; 4 TIL311's are not necessary to make the display work. But why not wire all four sockets, even if you won't pop for the displays right away? Leaving an empty socket won't hurt anything, and you never know when you'll run across a guy selling the displays for a buck apiece.

This circuit also illustrates one use of the two state code lines, SC0 and SC1. These lines are used to tell the hardware what sort of machine cycle the computer is currently going through. They are also useful in interrupt and DMA processing. The Pixie graphics chip uses them to help generate the video sync pulses, for example. A careful look at the SC0/SC1 timing diagram can help a lot if you're trying to use interrupt or DMA I/O. They're the only way to unambiguously tell what sort of machine cycle is going on at any particular time.

COSMAC PRICE BREAKTHRU!--In the year since I started putting this booklet together, RCA and Hughes have released plastic-package "consumer" versions of several chips in the COSMAC family. At this time I have only seen them available from Anacrona, but they should show up elsewhere at any time. The 1802 itself is only thirteen bucks, and the 1852 I/O port is only $1.85!! Now there is no longer any excuse not to use the 1802 family with the 1802 CPU. RCA's application notes make designing with the family as easy as piling building blocks atop one another. Good show.

STUDIO II CONVERSION--Another weird recent happening was Radio Shack's buyup of all RCA's remaining Studio II TV games. Just before Christmas 1978 every store was full of them, for $50. After Christmas they were all gone. I'll bet a box of TIL-311's that if you haunt the local garage sales, you'll eventually find a perfectly good Studio II for ten or fifteen bucks, once the family got bored with its rather slow, rather tame B&W TV games. An outfit called ARESCO (address in the back) puts out a conversion package for $5, which enables you to build a board which plugs into the game cartridge slot, enabling the Studio II to function as an ELF-type computer. I've seen it; I warrant you, it's worth the money if you have one of these items. The case and keyboards are super, and the RAM is CMOS CDP1822, which is a bloody expensive chip, plastic or otherwise. A full schematic and lots of other goodies are available from ARESCO. First class.

COSMAC'S FATAL FLAW--Those of you who haven't built an ELF yet ought to consider this; those who have probably know it already: COSMAC video graphics are slllllllowwwwww. This is a real head-scratcher, considering all the head-muscle RCA has at their command: Why tie the clock of an already slow (at 6 meg) machine down to a paltry 1.76 Mhz just for the sake of a video display generator? The 1861 uses TPA and TPB to generate sync pulses. Certainly, by adding a few flip-flops, they could have doubled (at least) the permissable clock speed to color burst. (Those nifty cheap 3.579 crystals, TV surplus) They lost their shirts on the Studio II because the games were, to be charitable, ponderous. One possible solution presents itself, now that COSMAC chips are so cheap: Build an ELF with two CPU's; one for processing at 4 megs or higher, and the other, which can be plastic, only handling display output to the TV screen. The hooker in this is setting up a screen buffer which both CPU's can access without fighting over the data bus or control lines. Otherwise it's no big deal for somebody with a little time and some RCA data sheets. Who's gonna do it?? (Added note: I recently learned that RCA had done exactly that: designed a computer to compete with the TRS-80, using two 1802's in such a configuration. Price was to be $500. It was called RECOMP-1, and it was pulled from the market at the last minute for mysterious reasons. Sunuvubitch. Those guys just can't seem to get their heads together about a real home computer using the 1802!!!)

A CRITICAL LOOK AT THE COSMAC VIP--In my continuing effort to spend myself backrupt, I bought a VIP, assembled, from Advanced Computer Products. I gotta say this for the place: seven days after the check was dropped in the mail, the computer landed on my doorstep. It's a purty machine, and the built-in keypad sure beats the bluejeans off of toggle switches. It's not a hell of a lot like the ELF, really--there's an operating system on ROM located at hex 8000, and run with some hocus-pocus using control lines which I don't really like. Still, it's a very good op system, particularly for the casette interface. VIP means Video Interface Processor, and that is just what it is: everything the VIP does is tied to the 1861 display system. This eliminates any need for LED displays, with a couple of exceptions, but it ties you down to that silly-ass 1.764 Mhz clock.

The VIP uses 2114 RAM, which enables you to get up to 4K pretty easily and cheaply. It's shipped with only 2K, but that's plenty for awhile. One gripe I have is that the Q line is tied internally to a little electromechanical honker. This honker is irritatingly squeaky and drove me nuts until I installed a little toggle switch on the board-cover to turn it on or off as desired. There are two 22/44 pin edge connectors on top of the VIP, enabling you to plug in either RCA's own accessory boards, or else boards you wrap yourself on Vector cards. RCA's boards are rather expensive, though they are a pretty royal blue. The only one I have is the Super Sound board, which has its own little language, PIN-8. That means Play It Now. I'd say Play It Eventually was more like it; coding music into the tables takes a little practice and quite a bit of time. For all that, it's worth it. Sound is good, and after only two weeks of suffering I got it to play Bach's "little" fugue in G minor, which drew applause from my wife. Super Sound uses the interrupt line, so you can't run the display simultaneously. Shame. I had the worst urge to animate an image of Bach turning over in his grave.

For all its flaws, the VIP is probably worth the money. They have an EPROM programmer for it which I intend to get eventually; intelligent EPROM burners cost a lot more than VIPs. The worst thing about the VIP is something that can be said of the ELF-II from Netronics or Quest's Super ELF: If you don't wire wrap it yourself, you won't learn as much. What are you doing this for? If you want to learn microcomputer hardware and software without going broke, the Popular Electronics ELF has no equal. If you just want to get into software, one of the others is just as good, and may be better. At this point the best of the commercial 1802 machines is the Netronics ELF-II. They've written an editor/assembler and tiny BASIC (by Tom PIttman, no less) and have lots of hardware add-ons. RCA seems to ignore the fact that their VIP could, in fact, be a lot more than just a game-hacker. I look upon it as a great potential COSMAC development system, and that is what I will continue to use it for.

AN EPROM CARD FOR THE VIP--The 2716 EPROM is a nasty exception to our beloved law that big-money IC's grow cheaper as time passes. The damned things still cost sixty bucks--when they're in stock. You can find the triple-supply version for forty bucks here and there, but the 1802 is a single-supply machine, and I gag on building two whole power supplies for the benefit of a single IC. One solution is to take a look at the 2758 EPROM. This is a single-supply version of the 2708, arranged as 1024 X 8 bits, which is to say plenty of room. How many 1K programs have <u>you</u> written recently??

The circuit below will plug directly into the expansion socket of the VIP. It will work just as well with an ELF, except that you have to give the ELF a way to disable RAM memory. I'll explain that shortly.

This is a good example of a relocatable ROM block, and about as simple a one as you'll find, so this might be a good time to give you folks who haven't met such a beast yet a little tutorial on how they work. The 4042 latch saves memory address bits A8 through A11 at TPA time. A8 & A9 are applied to the 2758's A8 & A9 inputs, and are actually used to select bytes on the chip. (Those two taken with A0 through A7 make ten address lines, and 2 to the 10th power is 1024.) A10 & A11, on the other hand, are used to indicate what

OEM's!

We've got 'em!

You know where to find the exciting, innovative parts your design needs – from the Electronic Components Division of TickTockTech! And the newest parts in the industry are also the fastest, because they're available *now* from your distributor. We've got 'em – come get 'em!

ZRF Connectors
A step beyond ZIF connectors. Zero Retention Force connectors do away with time-consuming levers or slides and completely eliminate component problems due to current flow!

Full-Pass Optical Couplers
Ordinary opto-isolators accept up to 2000 volts on the input side and deliver a meager 5 volts on the output. Why settle for this kind of signal attenuation? With Full-Pass, a 2000 volt input gives you 2000 volts output direct to your circuit.

16K VRAM
Very Random Access Memory lowers your chip count by simplifying addressing. Since data is stored at a totally random location, even one address line gives you as much precision as sixteen. Reading is simpler too, since data at any location is just as likely to be right as any other.

KB6650 Toaster Controller
A complete toaster control circuit on a single chip. Available 1st quarter '82.

XC880 Ultra-High-Speed Microcontroller
At 32 MHz, the fastest one-chip microcomputer in the industry! 92 instructions, including a convenient selection of 88 No-Ops.

Two-Way Zeners
A unique passive element. Two-Way Zeners operate in their breakdown regions in both directions, offering unimpeded current flow at a low price.

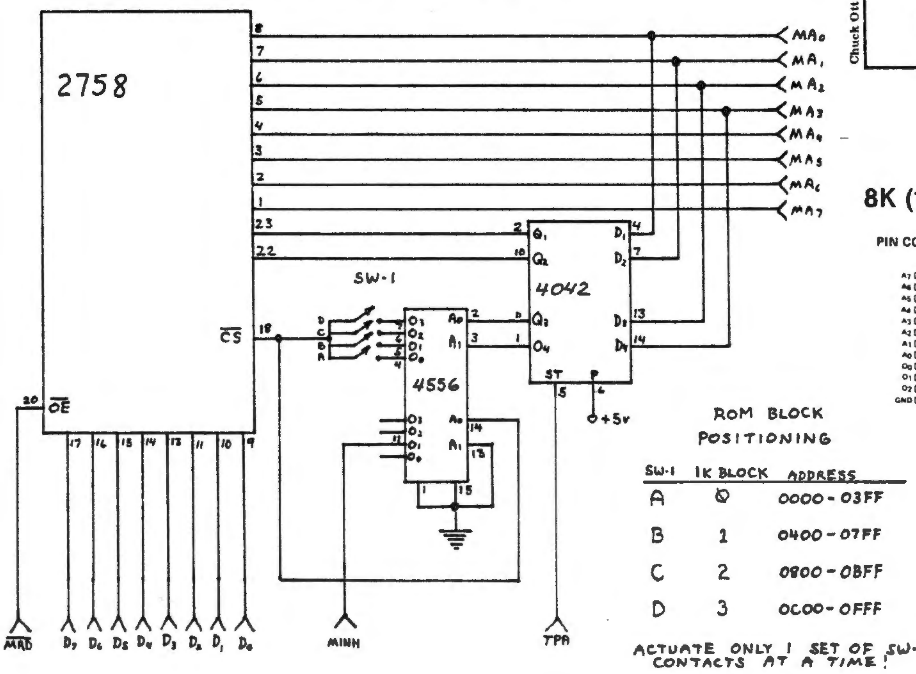
Another Innovation From

TickTockTech

Up-to-the-Minute Technology!

Chuck Ott

ELF / VIP EPROM Card



ROM BLOCK POSITIONING

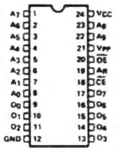
SW-1	1K BLOCK	ADDRESS
A	Ø	0000-03FF
B	1	0400-07FF
C	2	0800-0BFF
D	3	0C00-0FFF

ACTUATE ONLY 1 SET OF SW-1 CONTACTS AT A TIME!

VIP 2758 EPROM CARD 2/20/79

2758*
8K (1K × 8) UV ERASABLE PROM

PIN CONFIGURATION*



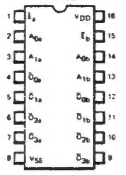
PIN NAMES

A_0-A_9	ADDRESSES
$\overline{CE}$/PGM	CHIP ENABLE/PROGRAM
$\overline{OE}$	OUTPUT ENABLE
O_0-O_7	OUTPUTS
A_R	SELECT REFERENCE INPUT LEVEL

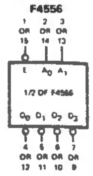
MODE SELECTION

PINS / MODE	$\overline{CE}$/PGM (18)	A_R (19)	$\overline{OE}$ (20)	V_{PP} (21)	V_{CC} (24)	OUTPUTS (9-11, 13-17)
Read	V_{IL}	V_{IL}	V_{IL}	+5	+5	D_{OUT}
Standby	V_{IH}	V_{IL}	Don't Care	+5	+5	High Z
Program	Pulsed V_{IL} to V_{IH}	V_{IL}	V_{IH}	+25	+5	D_{IN}
Program Verify	V_{IL}	V_{IL}	V_{IL}	+25	+5	D_{OUT}
Program Inhibit	V_{IL}	V_{IL}	V_{IH}	+25	+5	High Z

F4556



F4556



4556

single 1K block of memory the 2758 represents. In this circuit, you can set up SW-1 to "locate" the 2758 in any of the 1K blocks in the first 4K of 1802 memory space.

When the 2758's CS line is low, the chip is selected and can be addressed to fetch a byte from inside. If CS is high, the chip will not respond to any other control signals. The D0-D7 outputs remain in high-impedance tri-state, which is as good as having the chip "cut out" of the memory system. The 2758 must remain in this state until the 1802 sends outt the right 2-bit pattern on A10-A11. The 4556 is a 2-line to 4-line decoder. At any given time one of the 4 output lines is low. The others remain high. SW-1 allows one of these output lines to give that necessary low signal to the CS input. One of those output lines coresponds to one of the four possible combinations of high-low states on A10-A11. For example, when A10 & A11 are both low (0), output 0 (pin 4) of the 4556 goes low. If SW-1 contacts A are closed, the 2758 will be selected, and will respond to memory addresses 0000-03FF. Each of the four SW-1 contacts places the 2758 in a different range of memory, indicated by the table beside the circuit.

So much for selecting the 2758. In order to avoid a "fight" between the 2758's output lines and the output lines of the restt of the system, you must simultaneously deselect ("turn off") the rest of the memory system. The VIP conveniently gives you a line called MINH. If this line is brought high, VIP internal memory is deselected. Snag: you'd like to do both selecting and deselecting with the same signal, but the polarities required are not the same. Therefore, you have to invert the signal which selects the 2758 before it will deselect internal memory. I was tricky and avoided putting another chip on the board by using the second half of the 4556 as an inverter. I won't try to explain how this is done, but if you have a full 4556 data sheet you can follow it along from the truth table. It'd be good practice, so do it anyway.

The way this decoding scheme works, the delays in the 4556 allow both the 2758 and internal memory to be selected at once. A fight? No, because the 2758 has another pin, OE (output enable) which must be brought low before the 2758's outputs come down from their high-impedence limbo. The 1802's signal MRD does this at the proper time, and everyone is happy.

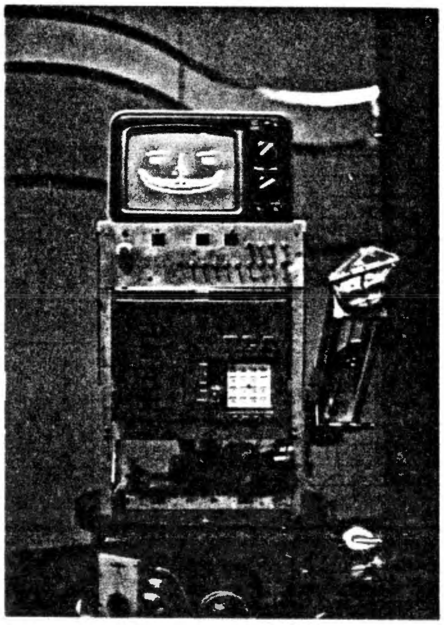
If you have an ELF wired as in the PE article, hooking this card to it is even simpler. Connect both the 2101s' pin 17s together, but separate them from the 5V supply. Now connect the line directly from the 2758's CS to both those pin 17's, and you're home free. You don't even need to use the other half of the 4556 as an inverter, because the 2101's have 2 select inputs, neither of which are used in the PE circuit. If pins 17 go low, the 2101's are deselected. Or, if the pins 19 are brought high, the same thing happens. Using pin 17 allows us to use the same signal which selects the 2758 to deselct the 2101's.

I've gone on at length here so beginners can get a feel for memory system juggling. The idea is to have only one set of outputs selected at any one time. You start with the required select polarity on a memory chip's chip select pin, and work back toward the CPU with whatever logic it takes to present the chip select pin with the state it needs to turn that chip on when its address comes up on the address bus.

Be careful when you design memory systems like this. If two sets of outputs get selected at once, the "fight" which results will overheat both chips and possibly ruin them. They may take the 1802 with them to CMOS heaven, so it could be an expensive mistake. If ever you run a memory system for the first time and it seems to "lock up" (another good reason to use address displays) cut power immediately and take another long, hard look at your circuit diagram.

PROGRAMMING EPROMS--No, I don't have a magic circuit built and debugged to program your 2758s for you. I get them programmed at work, and thus in my lazy way never pursued the issue. You have to admit, a $12,000 industrial EPROM burner sure beats the pants off a perfboard kluge! A $100 add-on board is available from RCA for the VIP which will do the job. I've not seen the circuit, so I can't say if it will work with an ELF. If anyone knows, by all means tell me; I've been waffling as to whether to order that board. The algorithm for programming 2758's is simple, however. I won't repeat it here; get the Intel data book and read up on it. My guess is that you could use a 555 as a 50 millisecond one shot to "stretch" a machine cycle to 50 milliseconds by bringing the WAIT line low with the desired byte on the data bus and selected on the address bus. If you hold data on the pins for 50 ms while applying a pulse to a program pin and 25 volts to another pin, the byte will be stored. There's more to it than that, and if you ever get it working I'd love to hear about it. Nuff said for now.

COSMO'S FACE--I take that back; there is something that the VIP is very good at: Giving my robot a face. For awhile I've been tinkering with a clanking heap of surplus submarine parts and wheelchair motors named Cosmo Klein. The Klein is an obscure mathematical allusion to the Klein Bottle, whos insides are identical to its outsides. Cosmo is a little like that, especially when he tips over and sends his insides spilling out onto the floor. Well, I got the notion that a COSMAC-generated face would be a marvelously humanizing touch. And so it is. If you want to see a good color picture of Cosmo and my VIP (with my own idiotically grinning mug in the background) check out Look Magazine dated April 30, 1979; it's the one with Jane Fonda on the cover. Maybe your library has it. The program which generates the face is included in this book, so I won't describe it here. Though you can't see it, my ELF is also inside, vainly trying to keep the monster from falling on his face. A CMOS robot is an old dream of mine, and I'm working on it, but for now I must pronounce his control circuitry (save for his face) a failure. Now you know who Captain Cosmo is. Yes indeed, that cute cartoon on the cover has a real model. Maybe in a future issue I'll have a little more to say about his inner workings. Once they work.

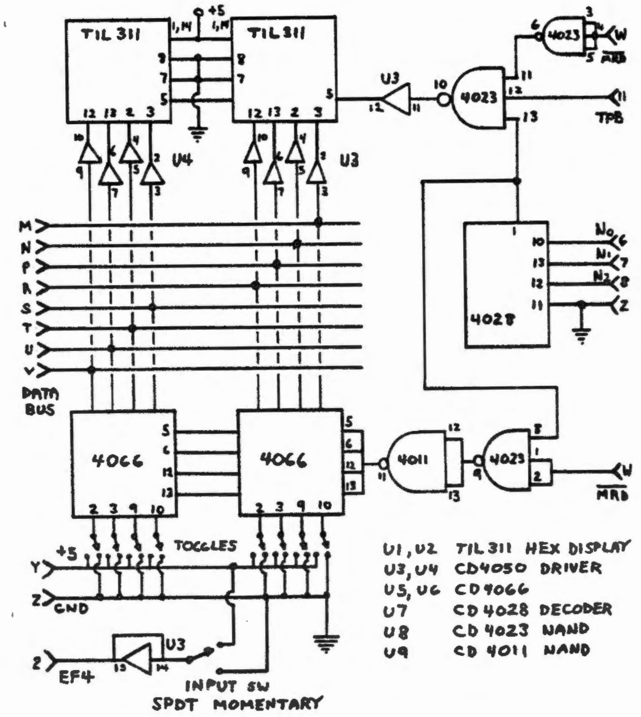


Cosmo the Robot with his access plates removed, so that you can see the mess inside. An ELF is hiding above and behind the VIP. On the right is Cosmo's arm. Not long after Bill Colsher took this photo, Cosmo tried to squeeze a Schlitz and stripped out his finger drive motor, proving that it takes a robot to crush his bare hand with a beer can.

ZOUNDS! by Jim Lisowski
for the basic ELF with 1 memory page
(multi-page systems will require changes!)

```
MA  OPCODE                COMMENTS
00    F8    Make location FF a work area
01    FF
02    AF
03    EF    X=RF
04    6C    Read byte from toggles (or keypad on ELF-II)
05    64    Display byte just read
06    2F    Compensate for '64' increment of RF
07    A2    Store toggle byte in R2.0
08    7A    Turn Q line off
09    82    Put contents of R2.0 in D
0A    A1    Store D into R1.0
0B    30    Branch to the custom delay routine
0C    1A
0D    21    Decrement R1
0E    81    Put contents of R1.0 into D
0F    3A    If D=0, go to the custom delay routine
10    0B
11    31    Otherwise, if Q is on, go turn it off
12    08
13    7B    But if Q is off, turn it on
14    22    Decrement R2
15    82    Put contents of R2.0 into D
16    32    If D=0 we are all done, so start over again
17    00
18    30    If not done, go to store and custom delay
19    09       routine again
1A    ..    Custom delay routine goes here
      ..
      ..
      ..
      30    Always end routine with:  30 0D
      0D
```

```
1. Routine: 21 21 21 21 30 0D
     Enter: 05 04 23

2. Routine: FD 20 32 0F 30 0D
     Enter: 77 0E 0F 10 13 30 33 44 or almost any number!

3. Routine: 76 FD 22 32 0F 11 11 30 0D
     Enter: FF 1F 33 11

4. Routine: 7E FD 22 32 0F 7E 11 11 30 0D
     Enter: 0B 11 12 13 15 16 19 18
```



ZOUNDS!!--Jim Lisowski gave me a marvelous little sound-effects routine which he wrote for his ELF-II; however, it will run as given on a basic ELF as well, as long as either machine is a single-page configuration. Jim did not initialize the high-order register halves, so altho the program might run on an expanded-memory system, it might not. Initializing high-order register bytes is no problem; the only hassle is pushing all the rest of the program down a few bytes with all the branching addresses correct. Really, you ELF hackers oughta be able to do that in your sleep, so I leave it to you.

Using it is easy. When entering the program into your ELF, decide which delay routine you want to use, and key it in. The delay routine must end with a branch to 00 0D, or the program will get hopelessly lost. When you run the program, toggle (or key on the ELF-II) bytes to change the nature of the sound effects.

This requires that the Q line be audio amplified to provide the sound. You can plug it into your stereo system (use the high impedance phono input) or, better still, wire up a little audio amp right on the ELF board. What I have is a little vector plugboard with an amp and speaker right on it. I anticipated VIP Simple Sound long before RCA announced it, but then again, the concept is pretty obvious. However you build it, it's nice to have available, and I think all ELF computers should have a VIP-compatible X-BUS socket so they can make use of VIP add-on hardware. More on that little item in a later subheading.

AN ELF ADAPTER BOARD FOR THE VIP--Just because the VIP has more bells & whistles than the ELF doesn't necessarily mean that the VIP can run ELF software. In fact, the VIP is sadly lacking in virtually all basic ELF I/O. About all they have in common is the Q lamp. I found it necessary and convenient to design a board for the VIP's X-BUS which would supply toggles, LED display, and INPUT switch to the VIP. This essentially gives you a 2K ELF with casette interface, Pixie mod, and other good stuff. Especially that casette interface, lack of which is the biggest single reason ELF programs remain small and rather trivial.

By design or by coincidence, the VIP and ELF do not use any of the same I/O codes, so adding ELF I/O to the VIP remained as easy as plugging in a board without further modifications to the VIP. I wire-wrapped the whole thing on a Vector 3662-5 plugboard. There was enough room after I was through to add the memory address display given earlier in this book, something else the VIP badly needs.

The devices on the board are accessed the same way and by the same codes and instructions ELF I/O is accessed. The only difference, really, is that the toggles can no longer be used to load programs via DMA as with the ELF's LOAD switch. (You won't really miss that too much now, will you?) The toggles can still be used to load programs from within the ETOPS operating system, which will, in fact, run on a VIP with this board.

A little more good news lies in wait if you have added the hex keypad to your ELF. The VIP hex keypad is driven in precisely the same way as the ELF keypad, by scanning the keyboard through a 4515 4-line to 16-line latching decoder. If you add this circuit board to your VIP, the VIP will also run ELF software which uses the ELF hex keypad. Even the I/O lines are the same (62 OUTPUT instruction and EF3). ELF software incorporating the Pixie mod for video will also run on the VIP, as the 1861 is turned on and off via the same I/O lines.

A word of warning here involving all add-on boards to <u>any</u> computer. Don't plug them in or yank them out with the power still on. You may not hurt anything--I've gotten away with it time and again--but you may also kill one or more of the IC's on the add-on board. A little rider on most CMOS IC data sheets tells you not to connect the circuit to a board under power--and they mean it.

VIP I/O LINE SUMMARY

EF1	Tests for video refresh done; don't use it!
EF2	Senses data from casette interface; don't use it!
EF3	Tests hex keyboard; OK except when using keyboard
EF4	Not used in VIP at all

I/O DATA STROBES

61	Turns 1861 off
62	Latches scan byte to hex keyboard
63	Strobes a byte to the 4508 output port
64	Used by op system to transfer control to code at M(00 00) but after that it's OK for user tasks
65	Not used by VIP
66	Not used by VIP
67	Not used by VIP
69	Turns 1861 on
6A	Not used by VIP
6B	Strobes a byte in from the 4508 input port
6C	Not used by VIP
6D	Not used by VIP
6E	Not used by VIP
6F	Not used by VIP

IN SEARCH OF COSMAC STANDARDS--Trying to run somebody else's software on your machine is usually a pain in the butt. Their I/O strobes strobe the wrong ports, and their call routines go looking for ROM in places where you have empty air. Etc. I can think, furthermore, of half a dozen small outfits and one large one (RCA) selling COSMAC hardware, and none of them are the least bit compatible to any other. Bad scene. The S100 nee Altair bus was helped along by the fact that the Altair, at its inception, was all there was. This makes de facto standards a lot easier to create.

We have no such luck. I have some suggestions. There are probably more ELF computers around than any other 1802 system, followed, I believe, by VIP and ELF-II in a dead heat. There isn't much in the original ELF to serve as a de facto standard, and most ELFs are subject to extensive jiggering by their makers. Given a choice between RCA and Netronics, I'll take RCA as the more likely of the two to last. And although I like the ELF II better than the VIP, I like the VIP's expansion slot arrangements better than the ELF II's. I propose that when you add expansion capabilities to your ELF, arrange them to accept hardware with the VIP X-BUS (expansion bus; the other slot I call the VIP IO-BUS) pin conventions.

Below I've included diagrams of the pinouts on the 2 VIP expansion slots, X-BUS & IO-BUS. IO-BUS is a much more limited pinout, but it has the advantage of using only one side of a 22/44 pin edge card. This makes it easier for home PC hackers to make their own boards; I have yet to make a 2-sided PC which I have been pleased with. X-BUS, while double-sided, lacks nothing I can discover, and if you're smart you'll home-hack your hardware on Vector wire-wrap cards, which are double-sided by nature.

Pin	Signal	Description
A	IN 0	8-bit input bus
B	IN 1	
C	IN 2	
D	IN 3	
E	IN 4	
F	IN 5	
H	IN 6	
J	IN 7	
K	$\overline{\text{INST}}$	Input byte strobe to latch U25
L	$\overline{\text{EF4}}$	Input flag line #4
M	OUT 0	8-bit output bus
N	OUT 1	
P	OUT 2	
R	OUT 3	
S	OUT 4	
T	OUT 5	
U	OUT 6	
V	OUT 7	
W	Q	Q flip-flop output line
X	EF3	Input flag line #3 (also used for hex keyboard)
Y	+5 V	Optional power for external logic
Z	GND	

Most of the inputs and outputs on X-BUS are self-explanatory; the bulk of them are taken right from the 1802 chip socket. I'll briefly discuss the unfamiliar signals so you know what to do with them.

Pin	Signal	Description
A	$\overline{\text{MWR}}$	Negative-going memory-write pulse
B	TPA	Early timing pulse for M address clocking, etc.
C	MA0	Memory address lines. High-order address byte appears on these lines during TPA time, followed by low-order address byte
D	MA1	
E	MA2	
F	MA3	
H	MA4	
J	MA5	
K	MA6	
L	MA7	
M	BUS 0	8-bit, 2-way tri-state data bus
N	BUS 1	
P	BUS 2	
R	BUS 3	
S	BUS 4	
T	BUS 5	
U	BUS 6	
V	BUS 7	
W	$\overline{\text{MRD}}$	Low for memory read machine cycles
X	CS	Chip select for operating system
Y	+5 V	Optional power for external logic
Z	GND	
1	CLOCK	CDP1802 clock output
2	$\overline{\text{EF4}}$	Flag input lines #3 and #4
3	$\overline{\text{EF3}}$	(Flag 3 also used for hex keyboard)
4	XTAL	Crystal frequency
5	$\overline{\text{EF1}}$	Flag input line #1
6	N0	Low-order 3 bits of N during 6N instruction
7	N1	
8	N2	
9	$\overline{\text{SPOT}}$	Video spot output
10	$\overline{\text{SYNC}}$	Video sync output
11	TPB	Timing pulse for clocking memory byte out, etc.
12	SC0	State code bit (+5 V for S1/S3, GND for S0/S2)
13	$\overline{\text{INTERRUPT}}$	Pulling to GND causes interrupt (22-KΩ input)
14	SC1	State code bit (+5 V for S2/S3, GND for S0/S1)
15	$\overline{\text{DMA-OUT}}$	Pull to GND for DMA-OUT cycles
16	Q	Q flip-flop output line
17	$\overline{\text{DMA-IN}}$	Pull to GND for DMA-IN cycles
18	RUN	+5 V when running, GND when RUN switch down
19	INDIS	Internal RAM-disable input
20	$\overline{\text{CDEF}}$	GND when RAM pages C, D, E, and F selected
21	+5 V	Optional power for external logic (same as Y-Z)
22	GND	

CS (Pin X) is an output. Whenever the operating system ROM is selected, this pin will be at +5V. Otherwise, this pin is at ground. Since your ELF has no op system ROM (tho using RCA's ROM is not a bad idea, if you can get one; see below) leave this pin unconnected. It is of minimal use anyway.

XTAL (Pin 4) in the VIP is the raw crystal oscillator output of 3.521280 Mhz. In the VIP this frequency is halved, and the 1802 runs at 1.76064 Mhz. The VIP, you see, does not use the 1802's on-chip oscillator. The reasoning (for the VIP) is that with one crystal they get a clock frequency for a CD-type chip (must be less than 2 Mhz) and the color-burst frequency (3.52 Mhz) for generating color video on the VIP Color Add-On Board. To use the VIP color board in an ELF, you will have to duplicate these conditions. You might as well use the same circuit as in the VIP, see below. I dislike using TTL in an all-CMOS system, but not all 74C00's will work in this mode. You may have to mess with different values for the resistors to make any particular 74C00 work. Some may not work at all.

I should mention that this circuit should look familiar to people who have added the Pixie graphics mod to their ELFs; it's the same one mentioned on page 43 of the July '77 PE. It's a good idea to use this mod if for no other reason than it allows you to use an el-cheapo (I've seen them for 49 cents) color burst crystal.

SPOT and SYNC, while exotic-sounding, are actually nothing more than pins 6 & 7 right from the CDP1861 socket. Sum them together with resistors and you have another composite video output. It can't hurt to connect them if your ELF has an 1861, tho they aren't of much use.

RUN comes right from pin 3 of the 1802. It's high when the VIP is running and low when not running, and it's controlled by the RUN switch, not by any 1802 internal logic. It's useful for clearing I/O ports and such. (Check out my use of this signal in the 1852 I/O circuit on Page 3.)

INDIS is an <u>input</u>; it is tied through a resistor to ground, where it will remain until some outside device pulls it to +5v. When that happens, all VIP internal memory will be disabled, meaning that all outputs will "float" harmlessly as open circuits, and read commands will be ignored. If you add an external memory board, you <u>must</u> make provision to disable internal memory, no matter if you have an ELF, VIP, or whatever. To make this work on an ELF, tie the pin 19's of all 2101 memory chips together and connect them to contact 19 (pure coincidence) of the X-BUS. Then (and don't skip this item) connect a 10K 1/4w resistor between X-BUS contact 19 and ground. If you omit the resistor, the chip enables of your 2101's will drift from ground to +5v, turning your memory chips on and off and raising havoc. This also leaves your memory chips open to damage by static electricity. Details count. If you're using the 2114 memory circuit shown on Page 2 of this book, things are a little trickier. You have to insert an OR gate between the 4081 and the chip select pins of the 2114's. The diagram below gives the necessary data. The end result is that regardless of what the output of the 4081 is doing, a high on the INDIS input to the OR gate will bring the 2114 chip selects high and disable the chips.

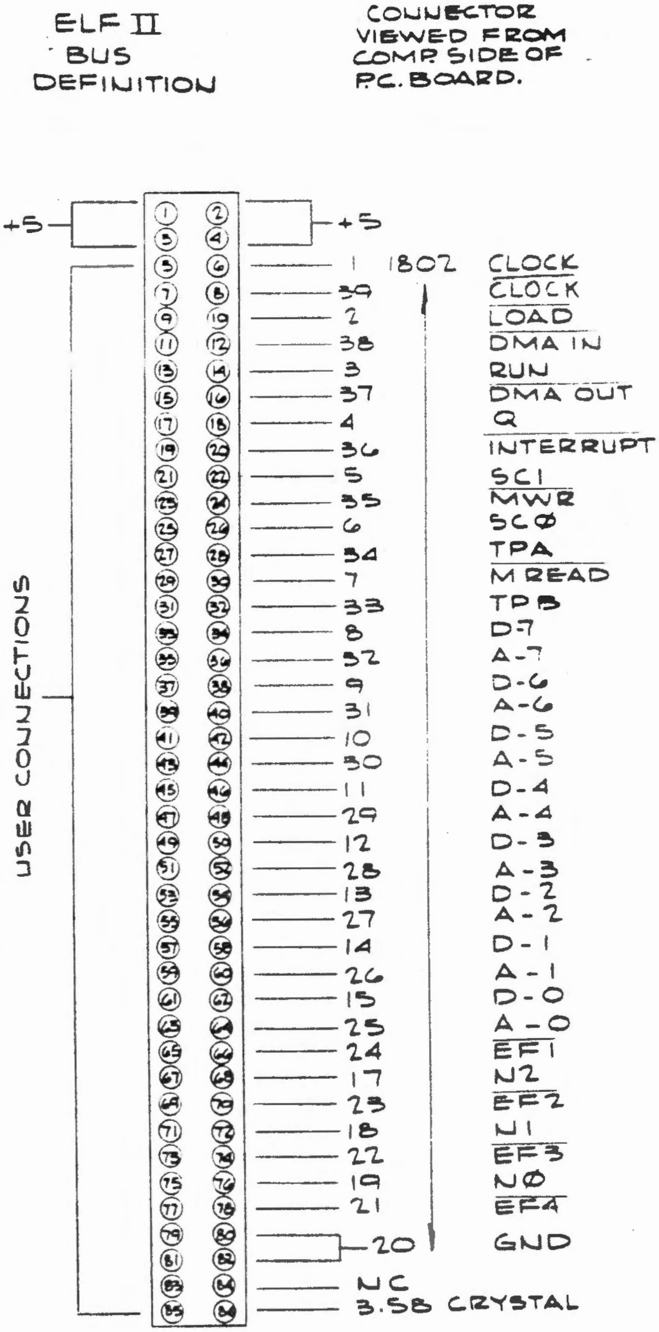
CDEF (X-BUS contact 20) is a weird one. Whenever VIP memory pages C,D,E, or F are selected, this pin goes to ground. The hooker is that only a tiny percentage of VIP's have those last four pages of internal memory. VIP's are sold with only 2K, which translates to pages 0 through 7. My hunch is that you had better leave this one disconnected. None of the VIP add-on hardware uses it, as far as I can tell.

So much for the X-BUS. Setting up an IO-BUS is easier. The only non-1802 contact is INST, contact K. INST is a latch strobe for the parallel input port in the VIP. When you bring this pin to ground, a byte is latched into the parallel input port from some external device. I personally don't care for that approach, as it puts the burden of generating the latch strobe on the external device. My 1852 I/O port arrangement automatically strobes the input port with TPB every machine cycle. This means, in essence, that any data which hangs around for at least one machine cycle will get latched into the port. If you prefer it the other way, it's no big thing to separate the input port pin 11 from TPB and strobe it externally. On the 1852, however, a high level pulse sets the latch. (The VIP uses 4508's for its parallel ports, with a different strobe polarity.) Tie 1852 pin 11 to ground through a 10K resistor and bring it to +5v to latch a byte in.

If you want to make your 1852 I/O system emulate the VIP's I/O ports (and that's a good idea) you have to strobe the input port with a 6B INPUT instruction. This translates to using pin 3 of the 4028 chip (all this refers to the 1852 I/O system, Page 3) Fortunately, the VIP strobes its output port with a 63 OUTPUT instruction, which also uses pin 3 of the 4028. On my diagram, replace the asterisk with a 3, and your 1852 I/O system will emulate the VIP's.

How far you carry VIP/ELF compatibility is a good question. Carry it far enough, and you no longer have an ELF, but a VIP. For that matter, it's entirely possible to lay hands on a VIP manual and wire-wrap your own VIP from scratch. The only "exotic" part is the op system ROM, and I have heard of people ordering it through RCA distributors; the part number is CDPR566. Everything else can be scrounged through normal channels. If any reader has made a VIP from scratch, I would like to hear about it.

I made the choice between the Netronics ELF-II and the VIP and I'll stick with it, even though Netronics has a whole lot more software available at this time. If you would prefer to configure your expansion slots to follow the ELF-II bus, I'm also including a description of its contacts. I don't like it because the connectors are expensive, hard to find, and somewhat fragile. But--take your choice. Since I've not played very much with the ELF-II, detailed description of the contacts is beyond me. You might get in touch with Jim Lisowski, my partner in Milwaukee, and if he reports enough interest we will cover the subject in a future volume of <u>Captain Cosmo</u>.



NETRONICS RESEARCH AND DEVELOPMENT LIMITED
333 LITCHFIELD ROAD (RTE. 202) NEW MILFORD, CONNECTICUT 06776
(203) 354-9375

SUPER SOUND REDUX--The last time I mentioned VIP Super Sound, I had had it less than a month and hadn't figured it out to my own satisfaction. Well, time makes many things clear. With a few months to ponder the device, I find I really like it. This is, of course, a little late to sell a tape of Super Sound Christmas music, but I have several recorded, and by Christmas '80 I may well have a good tape to hawk. Check with me.

GOOD KING WENCESLAUS

A NOTE TABLE

```
04 01      6F 6F 6F 71 6F 6F AA
        6C 6A 6C 6E AF AF 76 74
        7E 71 73 71 AF 6C 6A 6C
        6E AF AF 6A 6A 6C 6E 6F
        6F B1 76 74 73 71 AF B4
        EF E0
```

A MEASURE TABLE

```
03 00   01 05 08 0C 01 05 08 0C
        0E 12 15 19 1B 1F 22 26
        28 29 00
```

B NOTE TABLE

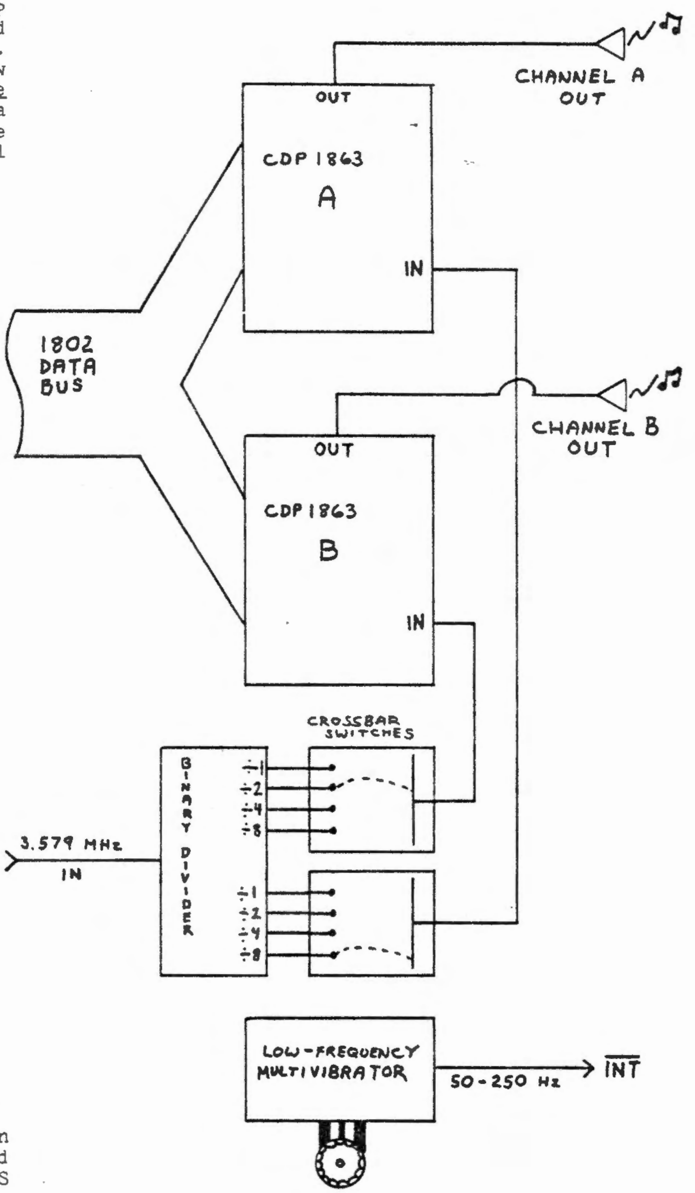
```
05 01      6F 71 73 74 76 74 B3
        94 33 71 36 34 73 71 AF
        73 74 B6 6F 71 B3 74 73
        B4 73 71 73 74 76 73 74
        71 73 7B BA 73 71 B6 B8
        B4 F3 E0
```

B MEASURE TABLE

```
03 80   01 05 08 0D 01 05 08 0D
        10 13 16 19 1D 21 24 27
        29 2A 00
```

BREAK TABLE

```
02 70   13 01 C0 16 01 E0 00
```



SUPER SOUND MADE SIMPLE

The VIP Super Sound Board is an interrupt-driven music generator. It's centered around the new CDP1863 chip, which is a CMOS programmable frequency divider. As best I can tell, the 1863 works like this: You strobe a hex byte into its latch and apply a relatively high frequency square wave to an input pin. The chip divides the input frequency by the hex byte and applies the result to an output pin. In Super Sound the input frequency is the 3.579 crystal frequency from the VIP clock. The output frequency can pretty obviously fall anywhere you want to put it, from subaudible to the low RF. This is not a for-sure description as I have not yet been able to lay mitts on an 1863 data sheet, but it's close.

Briefly, and in very general terms, the Super Sound hardware system works like this: (follow the bouncing diagram, if you will) The twin CDP1863's are programmable frequency dividers. You latch a hex byte into each one via the OUT instruction. Each 1863 will divide an input frequency by the hex byte latched into its registers. Each time you latch a different hex byte into one of the 1863's, a different tone will be produced by that channel.

You can also change the input frequency. One of the nifty things about Nature is the way musical octaves are related: Each octave is twice the frequency of the one beneath it, and half the frequency of the one above it. Sounds like binary to me, and by using a series of simple divide-by-two flipflops, Super Sound produces separate frequencies that are precisely one half, one fourth, and one eighth the input frequency. Those three taken with the unaltered input frequency make four input frequencies which can be applied to the 1863's--all harmonically related! That means that, for a given hex divider byte inside the 1863, the four input frequencies will produce the same note across four octaves.

One of the four input frequencies is selected by something called a crossbar switch; there is one such switch for each 1863. This device is the electrical equivalent of a 4-position rotary wafer switch. Instead of turning a knob, a binary code is strobed into the crossbar switch, which routes the input frequencies electronically.

Super Sound also has a multivibrator putting out pulses at a 50-250 Hz rate, adjustable by a little trimpot. The pulses strobe the interrupt line very much like the 1861 strobes the interrupt line to produce video refresh. (This is what we mean when we say a peripheral is "interrupt-driven.") Each time the pulser triggers an interrupt, PIN-8 checks to see if it's time to begin playing the next note or the next measure. That's all done in software; the board merely supplies the signals and timing. Spinning the trimpot changes the tempo, from ponderous Adagio to flitty Alegretto. Neat, huh?

There's somewhat more to it than that, but the rest is enhancement. PIN-8 has the capability to change the output envelope shape, giving you a fair approximation of flute, chimes, and other instruments. Also, the amplitude can be varied across sixteen levels. The effects, when used together, are subtle and surprisingly good.

The hardware is excellent. Interface is simpler than it sounds. There seems to be no hardware reason to keep you from plugging Super Sound into an ELF, assuming you had set up a 22/44 pin connector to look like the VIP X-BUS. (A good idea; I speak of it in detail elsewhere) Super Sound requires a 3.579 clock frequency and a memory inhibit input (no big deal) but otherwise needs nothing that an ELF cannot easily supply.

BUT....there's a helluva hook in it. Super Sound is helpless without the PIN-8 'language', and PIN-8 requires..ulp..1792 bytes of RAM. Unless I am far wrong, not one ELF in twenty has that much RAM. Most ELF-II and Super ELF machines can't handle it either, I think, though expansion is easier on them. Yes, PIN-8. Play It Now. Arrgh. Super Sound comes with a scratchpaddy little booklet virtually handwritten by the Great White ELF-Father, Joe Weisbecker. It looks very much like RCA gave him a budget of $1.98 to do the documentation. What's there is good, but it isn't very much...and then you come to PIN-8.

What you get is a hex dump of 1792 bytes, printed on what must be a 44- year-old Teletype with multiple sclerosis and leprosy of the ribbon. Gawdalmighty, RCA, surely you could give poor Joe a better source of hardcopy than that!! The E's frequently look like F's, and the D's often appear to be O's, and some of the digits are just downright hard to identify. I spent one very long evening tapping in the whole shebang. Naturally it didn't work. I was smart and saved it all to tape before I ran it, but even so...what the hell do you do when a 1792 byte program without a shred of source comment doesn't work? Easy...you verify the program in memory. That means you go blind a second time checking every last byte in RAM against that bleary listing. Save it to tape. It doesn't work. So...you do it again. And again. And again. Each time you weed out one more mistyped byte, until at last you've got it down and it runs.

Save yourself a lot of agony. Do it this way: Put an extra lamp on the table. Lay the booklet out nice and flat. Get two index cards, or pieces of scrap PC board, or aluminum sheet metal, or what have you. Lay them on the booklet so that only the first line of code is visible. Enter the code. Take a breath. Move the cards to expose the next line. Go back and verify the code. Do the same for the second line. And the third line. Next, take a break. Go upstairs and sweep the kitchen. Come downstairs again and do the next five lines, verifying the code as you go. Go upstairs and walk the dog. Do the next five lines. Go upstairs and scarf some lemonade and a few Triscuits. Do the next five lines. Go upstairs and read the mail. Do the next five lines. Go upstairs and make love to your wife/husband/girlfriend/boyfriend/whatever. Do the next five lines. Knock off for the day. Repeat procedure tomorrow. And the next day. And as many days as it takes. You'll have it right the first time.

It's not fun. I'd sell a tape with PIN-8 on it but I'm afraid RCA would sue me. So what can you do. Anyway, I should mention that Super Sound does not have its own speaker or amp; you have to plug it into your stereo system. Once it does work, and you run it, you'll hear a song I have to call the Aloha song. I don't know the words, but you'll recognize it. It's got a fair imitation of a twangy steel guitar and bouncy bass.

PIN-8 is not a language. It's a very large program with a lot of empty holes in it. You fill the holes with bytes calculated from sheet music, and when you flip the Run switch, it makes music. No commands or statements or anything else. When you save a song to tape, you save the whole shebang, PIN-8 and all, seven pages worth. There's wisdom in that, as anyone who has saved CHIP-8 only once and let his tape heads get magnetized will concur.

In conclusion of this subject, I'll say that PIN-8 is probably larger than it has to be. A COSMAC software wizard could probably write a "tiny" PIN-8 to fit in two or three RAM pages. You'd probably lose the steel guitar effects and maybe one channel, but it would still make music. I'd like to see Super Sound put up on an ELF. As Mr. Tolkien knew, Elves and music are meant for one another. I'd pay for such a program. Not much, maybe, but more than you'd get by giving it to an 1802 newsletter for free.

THE SOURCE WILL ALWAYS BE WITH YOU--I must not be a true programmer. I don't get a case of galloping paranoia when sombody wants to see my source code. I guess I'm just weird. Those of you whose experience is limited to the ELF may not know what this is all about. Well, in all but the simplest computers, programmers program in languages rather than raw opcodes like F8 93 BC 0E. Since computers only understand raw opcodes, a translator program converts an English language statement like ADD HOURS TO SUM into the raw opcodes which will accomplish the same job in the CPU. The English language statements are called the "source" code, and the resulting opcodes are called the "object" code.

You probably already know that a string of opcodes is tough to dope out if you didn't just write them ten minutes ago. That's why you put comments on the right side of your paper, to make them easier to understand. Informally, that's your "source"; it's the part of your program that makes the opcodes easier to understand.

The majority of programs you buy (excepting BASIC, which is, in a sense, all source code) you get only the object code, on a disk, tape, or on paper. No effort at all is made to explain how the thing works. In fact, some programmers burn a lot of mental calories trying to devise ways of keeping people from even displaying or copying the object code, all in the name of desperate paranoid secrecy.

Howcum? Not sure. Programming houses claim they don't want other programming houses to learn their techniques. That may be wise, as Sturgeon's law applies here; 90% of programming techniques are likely to be inefficient, sloppy, and full of klugework. If the consumer knew what he was getting, he wouldn't pay $150 for a goddam mess. Busnel baskets can cover sins as well as magic.

Individual hobbyist programmers have even less excuse. Damn few ever even try to make money off their programming skill, so passing out their techniques hurts not even themselves. They all fear that someone will rip off something that by their lack of initiative is worth nothing anyway. People can steal a program without the source. after all, and as I said above, most programming technique is miserable.

Ultimately, it must be a power thing. If only you have the source, you in effect control the program even after you sell it. Without the source, it's hard to change a program or custom-tailor it to specific needs. The man who has the source has the power. The majority of programmers, myself very definitely included, are ex-wimps and high school nurds, social misfits who were stepped on and laughed at through much of their development. It's human not to want to let go of power when you finally get it. Human. But paranoid. And needless. I double-dog dare all of you to quit being paranoid nurds and make your programs as easy for others to understand as they are for you. I dare you to match my promise: Never to publish a program without full source code and elaborate comments. My source will always be with you. Assume anybody else is hiding something.

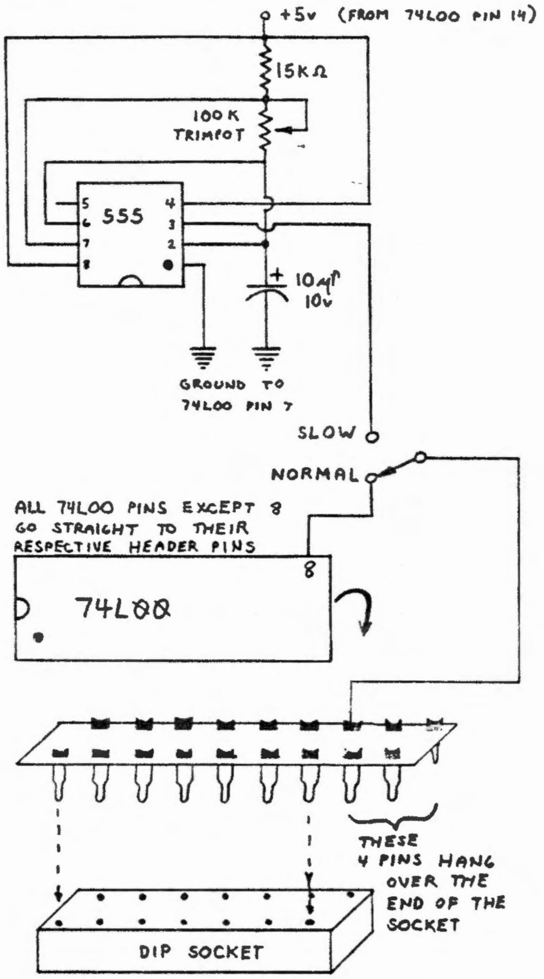
BINARY ARITHMETIC SUBROUTINE BOOK--RCA has a good-sized book devoted to a set of subroutines written for the COSMAC uP to enable it to handle binary and BCD arithmetic. Arithmetic is tough on the 1802, and a lot of junior ELF hackers lay out five beans for this book, only to find they can't use it. The Book is Manual MPM-206, "Binary Arithmetic Subroutines for RCA COSMAC Microprocessors." The subroutines give you the four arithmetic operations, using RF as a 16-bit accumulator. The arithmetic is done in binary, and there is a separate set of subroutines (also included) for converting BCD to binary and back. So far so good. The subroutines are written, however, using the Standard Call & Return Technique (SCRT) which I have discussed elsewhere in this book. I doubt one ELF owner in five understands SCRT, but worse yet, once you load the SCRT subroutines and arithmetic subrtouines and stack and whatnot into a 256 byte ELF, there's no room left for a program to use the subroutines. The subroutines use quite a few of the 1802's registers, so if you want to use both the subroutines and the registers, you have to custom-tailor SCRT to save any registers your own program wants to use, and I'm not even sure I can do that right now! For this book to be worth it, you must have: 1. An 1802 machine with at least 1K RAM; 2. An application that demands considerable number-crunching; and 3. Thorough understanding of SCRT, enough to reconfigure it to save any registers your application program intends to use. This is not trivial. (I tried it.) Save a fin. Grow a little before you buy it.

FAST/SLOW CLOCK GENERATOR--Jim Lisowski has come up with a terrific little geegaw that will work on any VIP or ELF-II computer. The 1802 family is totally static; that is, you can slow down the basic clock speed to one clock pulse every ten minutes, or even DC and stop things dead in their tracks. While it might seem dumb to have a ten second machine cycle, think again: You can actually watch the various control lines change state at such a clock speed. It's a helluvan education into the 1802 and uP theory generally. If you have one of those glomper clips with LED's for each pin, you can see TPA and TPB and the SC lines go on and off. Even a VOM will show things switching around. First rate.

What Jim put together is basically an adjustable 555 oscillator glued piggyback atop a 74L00 soldered to a 16 or 18 pin DIP header. A little slide switch switches the clock output from the 74L00 (full 3.579 Mhz) to the 555, which is adjusted by a little thumbwheel trimpot. How the thing goes together depends on what parts you decide to use, but mine turned into a very compact little item which you could probably use for a hockey puck without damage.

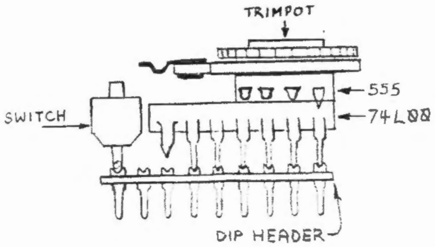
I built mine on an 18-pin DIP header to have a little extra room. I recommend doing this. You start by testing all the parts. I mean it! Once you glue something with Eastman 910, it takes God the Father with a jackhammer to get them apart. In particular, plug the 74L00 into the clock chip socket and make sure the computer runs with it. Some older 74L00's don't kick off under crystal control for some reason. Check the value of the resistor and (if possible) the cap. Run the 555 astable just to make sure. These are all good habits to get into, by the way. It makes troubleshooting a whole lot easier if you can assume the parts are all good.

Adjust the 74L00's pins so that they "grip" the contacts of the DIP header. Then take a needlenose and bend pin 8 straight outward. Solder the chip to the header so that pin 1 of the chip goes to pin 1 of the header, etc. This will leave you four unused header contacts behind the chip. Next, use your pliers and make all of the 555's pins point straight out. Put a touch of Eastman 910 on the pin 1 end of the 74L00, and glue the 555 atop the 74L00 so that pin 1 of the 555 is over pin 1 of the 74L00, etc.


+5v (FROM 74L00 PIN 14)
15KΩ
100K TRIMPOT
555
5
6
7
8
4
3
2
+ 10μf 10v
GROUND TO 74L00 PIN 7
SLOW
NORMAL
ALL 74L00 PINS EXCEPT 8 GO STRAIGHT TO THEIR RESPECTIVE HEADER PINS
8
74L00
THESE 4 PINS HANG OVER THE END OF THE SOCKET
DIP SOCKET

Drive a 2-speed clock home today!

At this point, take some wire-wrap wire and wire what you can without requiring the switch, resistor, pot, or cap. It takes small fingers, time, and skill to do this well. A small soldering iron with a needle tip also helps. Use as little heat as you can to avoid softening the plastic of the DIP header. Installing the switch comes next. I found a teeny-tiny one that actually fit between opposite contacts on the end of the DIP header. I soldered the outside switch contacts to the header and then glued the body of the switch to the end of the 74L00. Wire up what you can from the switch to the IC's. Now add the thumbwheel pot. I used the very common little one with a blue plastic wheel; Radio Shack sells them. It goes atop the 555, thumbwheel up (!) and contacts pointing back toward the switch. Eastman-910 it to the 555, and finish the wiring. The resistor fit well between the pot contacts and the switch; the cap had to be slung along the side. Put everything where it fits, and glue it all together. The reason for the glue is subtle: You're going to be pulling and pushing this thing into and out of an IC socket, and you'd prefer not to yank it apart if it gets stuck.



Operation is dirt-simple. The module replaces the clock generator IC. With the slide switch in one position, the output (pin 8) of the 74L00 will feed the clock line, and you'll have a 3.579 Mhz clock. Flip the switch, and the clock speed comes from the 555, depending on the setting of the pot.

On the ELF-II you can leave the module in place all the time. On the VIP the blue plastic cover won't close over it, so I had to make it easily removable. Sadly, it won't work on the original ELF configuration unless you've added the Pixie graphics mod, which is still another another reason to buy an 1861.

Obviously, the 555 will not make the VIP's video function properly, but it's fun to watch the contortions on the screen as you rev the old thumbwheel. Cheers!

INTERRUPTS MADE SIMPLE--Except for the Pixie video mod (which most people built without understanding a lick of it) damned few ELF hackers have made any use of the 1802's interrupt facility. Certainly the hardware is simple enough to use: a low pulse on the interrupt pin will give you an interrupt. Bang! But who knows exactly what an interrupt is, much less how to use one in a program?

At the bottom line, an interrupt is a tap on the CPU's shoulder, instructing it to drop what it is doing and take care of something else now. The best example I can supply is a hypothetical program (hypothetical because I haven't got the bugs out of it yet) for a buffered Morse Code keyboard. A Morse Code keyboard is a device on which you type ASCII characters and output Morse Code. Hams use them frequently, merrily typing away at a chirpy 120 wpm and copying the similar reply in their heads. A buffer their heads. A buffered keyboard is one in which typed characters are stored in a buffer while previous characters are being sent in Morse. (An unbuffered keyboard must complete each character before a new one is typed; such a program exists somewhere in this book.) It would seem, on the surface, that the CPU must be able to do two things at once: To output dots and dashes in proper Morse format, and simultaneously listen to the keyboard for new ASCII keypresses. Nay, nay. Enter the interrupt.

The program constantly monitors a 256-byte segment of memory, which we call a buffer. When the buffer is empty, the CPU keeps looping and checking. Let us say (without discussing the mechanism yet) that forty bytes are written into the buffer in rapid succession. The CPU immediately begins translating each ASCII byte into its Morse equivalent, and sending out the dots and dashes on the Q line. It has work to do for several seconds.

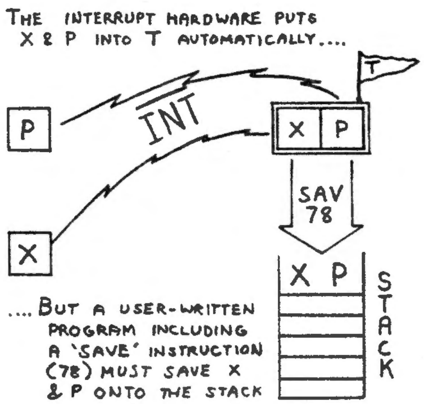
A second later, three more ASCII bytes are typed in. The processor is busily making dots and dashes, but the ASCII keyboard brings the interrupt lin low for a moment. WHAM! The CPU jumps to a routine which accepts the first ASCII byte and stores it on the top of the buffer. The whole process takes twenty microseconds or so. Then the CPU returns to complete a dot which is now twenty microseconds longer than dots usually are. That dot is not even complete when the second of the three bytes comes in and the interrupt line goes low. Another twenty microseconds to store the byte on the buffer, then the CPU finishes its dot. The third byte comes through during the space between the dot and the following dash. The space grows by twenty microseconds. I doubt anybody will notice.

As long as nothing is typed on the keyboard, the CPU will continue transmitting Morse characters until the buffer is once again empty. But as soon as a key is pressed, the interrupt is initiated and the key byte is stored in the buffer.

So much for the concept. How does the 1802 implement it? The interrupt line is checked by the CPU during the TPB pulse of every "execute" machine cycle. If during that TPB pulse the interrupt pin is low, the interrupt begins with the following machine cycle.

There is a flip-flop within the 1802 called the Interrupt Enable flip-flop. (IE, we will call it.) When the 1802 starts running, this flip-flop is set. When set, IE permits interrupts; when reset, the 1802 ignores the interrupt line. The first thing the 1802 does after it begins an interrupt is to reset IE. This prevents any more interrupts until IE is once again set by the program. Next, the 1802 stores the current value of X and P side by side as a single byte in an 8-bit cubbyhole called T. (Note that it is the number of the registers serving as X and P that are saved, not the 16-bit contents of those registers. That comes later, under program control.) Finally, the 1802 puts 1 in P and 2 in X. You had better have a workable interrupt routine pointed to by register 1, because the 1802 is going to start executing whatever code begins at M(R(1)). At this point the dedicated interrupt hardware inside the 1802 chip leaves off, and the work is taken up by the interrupt software, running with R1 as the program counter.

Implicit in the interrupt concept must be some way of going back to exactly what the "main" 1802 program was doing at the time the interrupt began. The interrupt hardware saved the "main" P and X in the T register. The interrupt program must now put that P and X somewhere. "Somewhere" means M(R(X)) by way of a SAVE instruction, and unless you change X first off, it will be M(R(2)) as set by the interrupt hardware.


THE INTERRUPT HARDWARE PUTS X & P INTO T AUTOMATICALLY....

....BUT A USER-WRITTEN PROGRAM INCLUDING A 'SAVE' INSTRUCTION (78) MUST SAVE X & P ONTO THE STACK

In a program that uses interrupts, R(2) should point to the top of a free area of memory at least five or six bytes in size. A convenient place is the very top few bytes of some memory page, that is, XX F0 - XX FF, with R(2) pointing to XX FF to start. This sixteen byte area is called the "stack." It's a stack because we can stack bytes up for storage in that area, and then flip them off the stack for re-use, much like a stack of dinner plates. (Yes, I realize that we stack the bytes downward from XX FF, but if it bothers you, flip your damned computer over.)

There's no reason this stack can't be used for other purposes in a program. Therefore, don't assume that M(R(2)) points to a free memory location. It may, in fact, point to some data needed by the "main" program. So, let us assume (and program accordingly) that any stack location below the "stack pointer" (R(2) here) is free, and any location above the stack pointer (between the stack pointer and XX FF) contains usable data. Any time you want to store a byte on the stack, DECREMENT R(2) one location first. Then, in this case, you would SAVE your T register onto the stack in that free location.

OK. Now we have X and P from the "main" program safely stored on the stack. We also will want to save the D register, certainly. That can go on the stack by decrementing R(2) again and then performing a STORE VIA R(2) instruction.

What else needs be stored? That depends. It is possible to save the DF flag by performing a RING SHIFT RIGHT instruction to get DF back into the D register, and then SAVE the D register as described above. Also, if your interrupt routine wants to use any of the same registers the "main" program is using, you have to save those as well. This is done using GET HIGH R(X) and GET LOW R(X) instructions to move the register halves into the D register, and then storing the D register as above. Remember to DECREMENT R(2) before you store something!!

As illustration, the following snippet will take care of the initial housekeeping for an interrupt routine that shares D, DF, and R(9) with the "main" program:

```
     MA   OPCODE

  XX 20    22        DEC R(2)
     21    78        SAVE T to M(R(2))
     22    22        DEC R(2)
     23    73        STORE VIA X & DECREMENT
     24    76        RING SHIFT RIGHT
     25    73        STORE VIA X & DECREMENT
     26    89        GET LOW R(9)
     27    73        STORE VIA X & DECREMENT
     28    99        GET HIGH R(9)
     29    52        STORE VIA R(2)
```

(Onward with the interrupt routine's real work.)

Notice that your stack has grown five bytes by this little operation, so you had better have at least that much (and preferably a good deal more) space available for the stack to grow.

This process can be extended to include any or all GP registers, but as you can see, saving any large number of registers would make for a very long interrupt routine that takes lots of memory space and execution time. If at all possible, assign certain registers to the "main" program and leave them alone in your interrupt routine. If your interrupt routine does not affect the DF flag, don't save it. Often you can get by simply by saving X & P from the T register, and then D. The video refresh routine for the Pixie mod does this.

Once the interrupt routine has done all the saving it has to do, it runs just like any other 1802 program, except that interrupts are inhibited by the reset state of the IE flag. Eventually you will want to get back to the serious business of the "main" program, so all that data that you saved on the stack has to be popped off again and plugged back into the CPU's inner workings.

Writing the restore section of the interrupt routine is a matter of doing everything again in reverse. Whereas a STORE VIA X & DECREMENT instruction stores the contents of the D register at M(R(X)) and then decrements R(X); the LOAD VIA X & ADVANCE instruction loads the contents of M(R(X)) into D and then increments R(X). In our example, start by restoring R(9). DF may be restored by loading its byte from the stack and then using a SHIFT LEFT instruction to shove the most significant bit back into DF. D is restored just by loading its byte from the stack into D.

The last step uses the RETURN instruction to take the stored X & P from their side-by-side byte and insert them back in the registers from whence they came. This will, in effect, "turn on" the "main" program and things will begun running agiain as though no interrupt had occurred. RETURN also sets the IE flag to 1, and enables further interrupts. However, the program counter for the interrupt routine, R(1), now points to the end of the interrupt routine rather than the beginning, so the next time an interrupt occurs, the 1802 will begin executing instructions or garbage that lie just beyond the end of the interrupt routine, and if experience is any guide the computer will go berserk. The trick is to perform a short branch back to the byte immediately preceding the true beginning of the interrupt routine, where your RETURN instruction should lie. Once the RETURN is executed, R(1) will be pointing right at the routine's start, ready to go for the next interrupt that comes down the line.

To recap, here is a model of a typical interrupt routine. In this case, it saves X & P and D (essential for all interrupt routines) as well as DF and R(9). (optional)

```
     MA   OPCODE

     1F    70        RETURN
  XX 20    22        DEC R(2)
     21    78        SAVE T to M(R(2))
     22    22        DEC R(2)
     23    73        STORE VIA X & DECREMENT
     24    76        RING SHIFT RIGHT
     25    73        STORE VIA X & DECREMENT
     26    89        GET LOW R(9)
     27    73        STORE VIA X & DECREMENT
     28    99        GET HIGH R(9)
     29    52        STORE VIA R(2)
```

(Code for real interrupt business here)

```
  XX 59    72        LOAD VIA X & ADVANCE
     5A    B9        PUT HIGH R(9)
     5B    72        LOAD VIA X & ADVANCE
     5C    A9        PUT LOW R(9)
     5D    72        LOAD VIA X & ADVANCE
     5E    FE        SHIFT LEFT
     5F    72        LOAD VIA X & ADVANCE
     60    30        SHORT BRANCH
     61    1F         (to XX 1F)
```

People who start playing with interrupts soon start to worry about a disturbing possibility: What happens if the interrupt line goes low soon after the 1802 begins running, before the 1802 has a chance to point R(1) to the interrupt routine? Only R(0) is cleared to 00 00 upon CPU reset! (Read that line again!) R(1) may contain any old garbage until you fill it with the interrupt routine address. If an interrupt occurs in the few dozen microseconds between RUN and the time you point R(1), the machine will probably go berserk. There's a trick to get around it. Consider the following 3-byte instruction sequence:

```
     E3      SET X = R(3)
     71      DISABLE
     53      (hex byte 53; not intended as a
              STORE VIA R(3) instruction!)
```

This sequence should be used in a program where P=3 and X=5. If it were used in a program where P=3 and X=2, the hex byte would by 23 and not 53. What we've done here is "fooled" the 1802 into reading its XP byte from the regular program sequence instead of the stack, by setting X=P in the first instruction. (Review the action of the DISABLE instruction here: It reads the byte from M(R(X)) and interprets it as XP, then loads the X & P registers with the XP it read. Since the XP you stored immediately after the DISABLE instruction is the XP that belongs in the program at that point, you've changed nothing--except the state of the interrupt enable flipflop, IE. Interrupts will be disabled until a similar sequence with a RETURN in place of the DISABLE is encountered. In this way, you can turn interrupts on and off at any time within the program, as required.

That special case of disabling interrupts immediately can be handled even more simply. At RUN time, P and X are both 0, so the SET X instruction is unnecessary. Loading the sequence: 71 00 as the first two bytes in memory will disable interrupts at RUN time immediately. Interrupts will be disabled until a three-byte RETURN sequence as described above is encountered.

Another caution is worth considering: If your interrupt line is held low for any long period of time (longer than a millisecond is a verrrrrrrry long time) your interrupt routine may run and return before the interrupt line goes high again. If that happens, you will immediately get another interrupt, and whatever job the interrupt routine is supposed to do will be done twice, or three times, or more. As a rule of thumb, make sure that your interrupt pulse to the interrupt line is shorter than the execution time of the interrupt routine. It's relatively easy to configure a 555 timer as a short-interval one-shot, which means that you feed it a long pulse and get a short pulse out.

An Animated Video Face System

This program puts an animated face on a TV screen. Next Halloween, mount your CRT in the window instead of a pumpkin. (CRTs don't rot, and candles are hardly state-of-the-art...) It will goggle, glower, and grin at passing children until one of them puts a brick through your window and CRT. Or, if you're not into Halloween, mount a CRT and a VIP atop your robot, and it will look a good deal dumber than it may in fact be. But it will be a lot more popular at parties.

Whatever you decide to do with it, this is how it works: When the program is run, debris from previous programs is erased from the screen. SMLCPY copies a smiling face pattern to the display page from page 03. The smile as stored has no eyes. Eyes are written in immediately after the smile is copied. That done, subroutine EF4TST is called. This samples flag line EF4. If EF4 is low, 01 will be stored in the memory location where CHIP-8 stores variable D. If EF4 is high (normal condition) 00 will be stored in variable D. Immediately after returning to the MAIN program, VD is tested. If it is found to be 01, CHIP-8 subroutine CHPLCK is called. CHPLCK scans a tongue-shaped pattern scross the smiling face's upper lip about three-fourths of the way, reverses and goes back to its starting point, then vanishes. During the time the tongue is scanning, closed eyes are displayed on the smiling face.

If CHPLCK was called, the program returns to the beginning and starts again. If not, a timer is begun, and nothing further happens for about three fourths of a second. When the timer decrements down to 00, a random number from 00 to OF is pulled. If this number was 00, FRNCPY is called, which copies a whole new face (also minus eyes) into the display page. This is a frowning face, and looks rather as though the creature was just forced to sample a mouse-entrails-on-rye sandwich. Eyes are immediately copied in, and the timer begun again. Each time the timer reaches 00, I is incremented by four and the eyes are recopied. This means the eyes change every three fourths of a second, and follow a definite sequence of rolling, blinking, and looking around. The frowning face remains on the screen for four countdowns of the timer, and thus for four eye-changes. After that time the smile is copied back into the display page, and the whole sequence is begun again.

There is a CHIP-8 subroutine called PGPEEK which will permit you to peek at any page in memory. This is strictly a bonus and is not used in normal execution of the program. To use PGPEEK, replace the CHIP-8 instruction at 0200 with 1550. This will jump immediately to PGPEEK. PGPEEK waits for a keypad key pressed, then displays the memory page corresponding to the number pressed. If you press a number for which no memory page is socketed (say, page 0C in a 2K VIP) the display will be all white. PGPEEK does not depend on any other subroutine in the system and can be used in other VIP programs.

The only extra hardware required by this system is some means for pulling EF4 low to signal the chop-licking subroutine. This does not need to be bounceless at all; in fact, touching a wire to a ground pin works just fine. I used this method to be able to trigger the tongue by remote control at appropriate times, say, when a suitably scrumptious woman entered the room. Most women feel silly slapping a TV set, and some might even be amused, since TV sets have no legs and are considered "safe." For Halloween applications you might want to replace subroutine EF4TST with a CHIP-8 subroutine for pulling and testing a random number to determine whether or not to trigger CHPLCK. This particular subroutine, stored at the location where EF4TST is now, will do the job well:

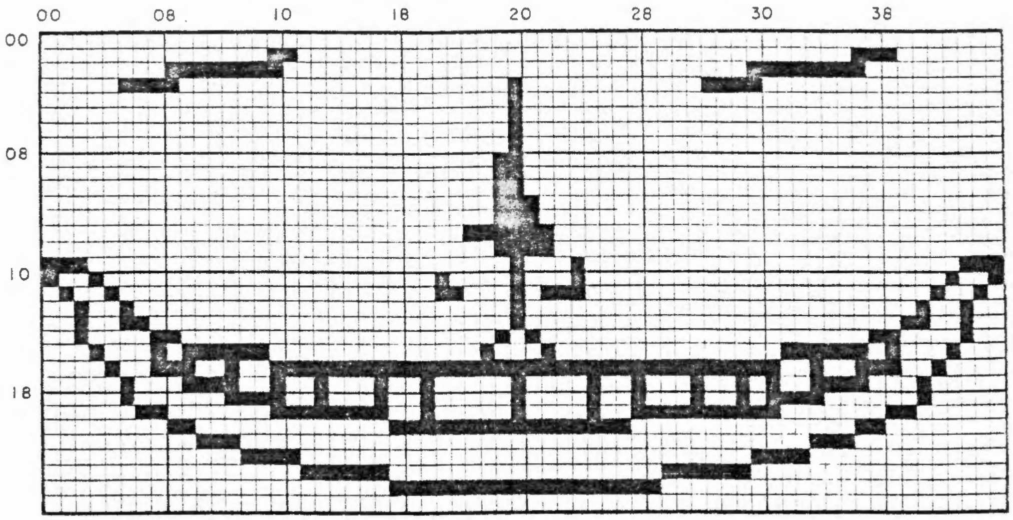
```
06 20   C00F    Pull random from 00 - 0F
        4005    Skip if V0 ≠ 05
        6D01    VD = 01
        00EE    Return to MAIN
```

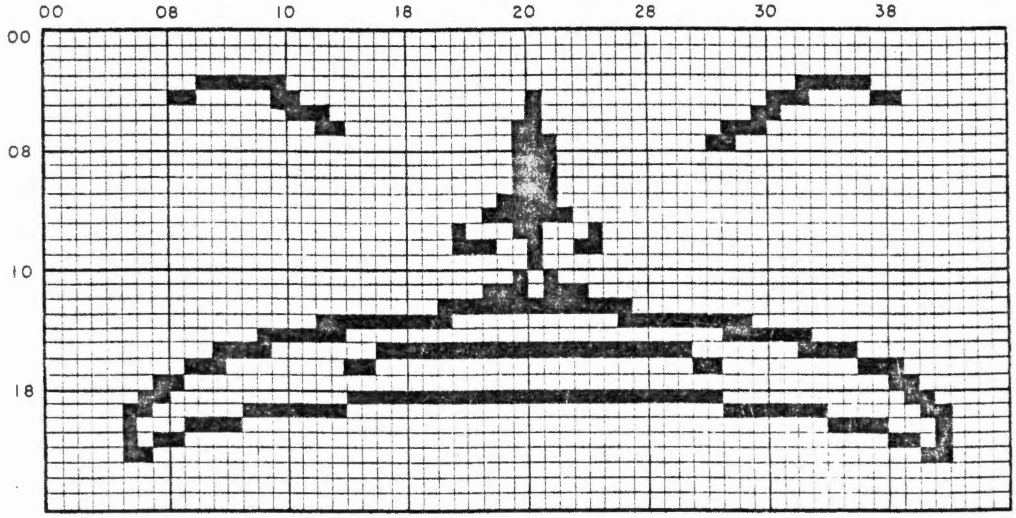
ANIMATED FACE CONTROL SYSTEM PAGE 02
MAIN PROGRAM

```
 MA     OPCODE   MAIN

02 00   00E0     ERASE ENTIRE SCREEN
   02   0600     DO SUB: SMLCPY
   04   6108     V1 = 08
   06   6206     V2 = 06
   08   6330     V3 = 30
   0A   6400     V4 = 00
   0C   6800     V8 = 00
   0E   6900     V9 = 00
   10   6C04     VC = 04
   12   A290     POINT I TO FIRST EYE
   14   D124     SHOW LEFT EYE
   16   D324     SHOW RIGHT EYE
   18   0620     DO SUB: EF4TST
   1A   3900     SKIP IF V9 = 00
   1C   1226     GO TO TIME DELAY
   1E   4D01     SKIP IF VD ≠ 01
   20   2500     DO SUB: CHPLCK
   22   4D01     SKIP IF VD ≠ 01
   24   1202     GO TO DO SMLCPY
   26   6520     V5 = 20
   28   F515     V5 = TIMER CONSTANT
   2A   F607     V6 = CURRENT TIMER VALUE
   2C   3600     SKIP IF V6 = 00
   2E   122A     GO TO 022A
   30   4804     SKIP IF V8 = 04
   32   1200     GO TO START
   34   C00F     V0 = RANDOM FROM 0 - F
   36   3901     SKIP IF V9 = 01
   38   3000     SKIP IF V0 = 00
   3A   1246     GO TO 0246
   3C   0650     DO SUB: FRNCPY
   3E   D124     SHOW LEFT EYE
   40   D324     SHOW RIGHT EYE
   42   6800     V8 = 00
   44   6901     V9 = 01
   46   3900     SKIP IF V9 = 00
   48   7801     V8 = V8+01
   4A   D124     SHOW LEFT EYE
   4C   D324     SHOW RIGHT EYE
   4E   FC1E     I = I+VC  (04)
   50   7401     V4 = V4+01
   52   3407     SKIP IF V4 = 07
   54   1214     GO TO SHOW EYES
   56   A290     POINT I TO FIRST EYE
   58   6400     V4 = 00
   5A   1214     GO TO SHOW EYES
```

ANIMATED FACE CONTROL SYSTEM PAGE 02
EYE PATTERN STORAGE

```
 MA     DATA

02 90   3E 38 38 3E
   94   3E 32 32 3E
   98   3E 26 26 3E
   9C   3E 0E 0E 3E
   A0   3E 26 26 3E
   A4   3E 22 22 3E
   A8   3E 32 32 3E
   AC   3E 38 38 3E
   B0   3E 32 32 3E
   B4   3E 26 26 3E
   B8   00 3E 26 26
   BC   00 00 3E 2A
   C0   00 3E 0E 0E
   C4   3E 0E 0E 3E
   C8   0E 0E 3E 3E
   CC   26 26 3E 3E
   D0   32 32 3E 3E
   D4   38 38 3E 3E
```





```
ANIMATED FACE CONTROL SYSTEM PAGE 03
SMILING FACE PATTERN STORAGE

 MA        DATA

03 00      00 00 00 00 00 00 00 00
   08      00 01 80 00 00 00 03 80
   10      00 FF 00 00 00 01 FE 00
   18      07 80 00 01 00 0F 00 00
   20      00 00 00 01 00 00 00 00
   28      00 00 00 01 00 00 00 00
   30      00 00 00 01 00 00 00 00
   38      00 00 00 01 00 00 00 00
   40      00 00 00 03 00 00 00 00
   48      00 00 00 03 00 00 00 00
   50      00 00 00 03 00 00 00 00
   58      00 00 00 03 80 00 00 00
   60      00 00 00 03 80 00 00 00
   68      00 00 00 0F C0 00 00 00
   70      00 00 00 03 C0 00 00 00
   78      E0 00 00 01 10 00 00 07
   80      90 00 00 21 10 00 00 09
   88      48 00 00 31 70 00 00 12
   90      24 00 00 01 00 00 00 24
   98      26 00 00 01 00 00 00 64
   A0      21 80 00 02 80 00 01 84
   A8      11 7E 00 04 40 00 7E 88
   B0      09 C9 FF FF FF FF 93 90
   B8      04 79 22 41 09 12 9E 20
   C0      04 0F 22 41 09 12 F0 20
   C8      03 01 FE 41 09 FF 80 C0
   D0      00 C0 01 FE FE 00 03 00
   D8      00 38 00 00 00 00 1C 00
   E0      00 07 80 00 00 01 E0 00
   E8      00 00 7E 00 00 7E 00 00
   F0      00 00 01 FF FF 80 00 00
   F8      00 00 00 00 00 00 00 00
```

```
ANIMATED FACE CONTROL SYSTEM PAGE 07
FROWNING FACE PATTERN STORAGE

 MA        DATA

07 00      00 00 00 00 00 00 00 00
   08      00 00 00 00 00 00 00 00
   10      00 00 00 00 00 00 00 00
   18      00 3F 00 00 00 00 3E 00
   20      00 C1 80 00 80 00 E1 80
   28      00 00 E0 00 80 01 80 00
   30      00 00 30 01 80 07 00 00
   38      00 00 00 01 C0 0C 00 00
   40      00 00 00 01 C0 00 00 00
   48      00 00 00 01 C0 00 00 00
   50      00 00 00 01 C0 00 00 00
   58      00 00 00 03 C0 00 00 00
   60      00 00 00 07 E0 00 00 00
   68      00 00 00 11 88 00 00 00
   70      00 00 00 1C 98 00 00 00
   78      00 00 00 00 80 00 00 00
   88      00 00 00 07 70 00 00 00
   90      00 00 00 3F FE 00 00 00
   98      00 00 3F E0 03 FE 00 00
   A0      00 03 F0 00 00 07 E0 00
   A8      00 1E 03 FF FF E0 3C 00
   B0      00 70 0C 00 00 18 03 80
   B8      01 80 00 00 00 00 00 C0
   C0      03 00 0F FF FF F8 00 60
   C8      06 07 F0 00 00 07 F0 30
   D0      04 78 00 00 00 00 0F 10
   D8      05 80 00 00 00 00 00 D0
   E0      06 00 00 00 00 00 00 30
   E8      00 00 00 00 00 00 00 00
   F0      00 00 00 00 00 00 00 00
   F8      00 00 00 00 00 00 00 00
```

If you decide to use this suoroutine, the call instruction at 02 18 will have to be changed to 2620, because the new subroutine is a CHIP-8 subroutine, and call processes are a little different. When in place, the chop-licking process will be initiated automatically, and no hardware need be added to the VIP.

Note that this program requires a 3K VIP. Remember, that highest RAM page is the display buffer, and the next to highest RAM page is a CHIP-8 playground and best to stay out of. If you have a 4K system, make these program changes:

```
05 57   0E
06 04   0F
06 21   0E
06 53   0F
```

These changes all involve display and variable-storage page pointers. If your VIP is some jacked-up super system with offcard RAM, I will leave the modifications to you.

And that's about it. Presumably, you could make up a whole range of face expressions and store each as a separate face in a page of RAM. Certainly there's room enough in the VIP for three or four more faces, and that's a lot. Animating the eyebrows would be a cinch if you wanted to take the time. Why not do it?

ANIMATED FACE CONTROL SYSTEM PAGE 06
MACHINE LANGUAGE SUBROUTINES

```
 MA     OPCODE        SUBROUTINE: SMLCPY

06 00   F8 03 BD      POINT RD.1 TO PAGE 03
   03   F8 0B BE      POINT RE.1 TO PAGE 0B
   06   F8 FF AD AE   POINT TO TOPS OF PAGES 03 & 0B
   0A   EE            X = E
   0B   0D            LOAD M(R(D)) INTO D
   0C   2D            DEC RD
   0D   73            STORE D AT M(R(E)) & DEC RE
   0E   8E            GET LOW BYTE RE.0
   0F   3A 0B         IF D = 00, GO TO M(06 0B)
   11   0D            LOAD BOTTOM BYTE OF PAGE 03
   12   5E            STORE D IN BOTTOM BYTE OF PAGE 0B
   13   D4            RETURN TO MAIN
```

```
 MA     OPCODE        SUBROUTINE: EF4TST

06 20   F8 0A BE      POINT RE TO RAM LOCATION WHERE
   23   F8 FD AE      CHIP-8 STORES VALUE OF VD
   26   3F 2C         IF EF4 IS LOW, GO TO M(06 2C)
   28   F8 01 5E D4   LOAD 01 INTO VD AND RETURN TO MAIN
   2C   F8 00 5E D4   LOAD 00 INTO VD AND RETURN TO MAIN
```

```
 MA     OPCODE        SUBROUTINE: IDEC9

06 40   2A 2A 2A      DECREMENT RA (WHERE CHIP-8 STORES
   43   2A 2A 2A      THE VALUE OF MEMORY POINTER I)
   46   2A 2A 2A      NINE TIMES
   49   D4            RETURN TO MAIN
```

```
 MA     OPCODE        SUBROUTINE: FRNCPY

06 50   F8 07 BD      POINT RD.1 TO PAGE 07
   53   F8 0B BE      POINT RE.1 TO PAGE 0B (DISPLAY)
   56   F8 FF AD AE   POINT TO TOPS OF PAGES 07 & 0B
   5A   EE            X = E
   5B   0D            LOAD M(R(D)) INTO D
   5C   2D            DEC RD
   5D   73            STORE D AT M(R(E)) & DEC RE
   5E   8E            GET LOW BYTE RE.0
   5F   3A 5B         IF D= 00, GO TO M(06 5B)
   61   0D            LOAD BOTTOM BYTE OF PAGE 07
   62   5E            STORE D AT BOTTOM BYTE OF PAGE 0B
   63   D4            RETURN TO MAIN
```

ANIMATED FACE CONTROL SYSTEM PAGE 05
CHIP-8 SUBROUTINES

```
 MA     OPCODE   SUBROUTINE: CHPLCK

05 00   D124     ERASE LEFT EYE
   02   D324     ERASE RIGHT EYE
   04   A2BC     POINT I TO CLOSED EYES
   06   D124     SHOW LEFT EYE CLOSED
   08   D324     SHOW RIGHT EYE CLOSED
   0A   6511     V5 = 11
   0C   6630     V6 = 30
   0E   6709     V7 = 09
   10   6709     V7 = 09
   12   6709     V7 = 09
   14   6A01     VA = 01
   16   A400     POINT I TO TONGUE
   18   D659     SHOW TONGUE
   1A   86A5     V6 = V6-VA
   1C   F71E     I = I+V7
   1E   361D     SKIP IF V6 = 1D
   20   1518     GO TO 0518
   22   86A4     V6 = V6+VA
   24   0640     DO SUB: IDEC9
   26   D659     SHOW TONGUE
   28   3630     SKIP IF V6 = 30
   2A   1522     GO TO 0522
   2C   A290     POINT I TO FIRST EYES
   2E   6400     V4 = 00
   30   D124     SHOW LEFT EYE
   32   D324     SHOW RIGHT EYE
   34   00EE     RETURN TO MAIN
```

```
 MA     OPCODE   SUBROUTINE: PGPEEK

05 50   F30A     WAIT FOR ANY KEY
   52   0556     DO SUB AT 0556
   54   1550     GO TO 0550
   56   F8       (SUBROUTINE TAKES NUMBER OF KEY
   57   0A        PRESSED FROM STORAGE LOCATION
   58   BD        OF V3 AND STORES THAT NUMBER
   59   F8        IN THE HIGH-ORDER HALF OF RB.
   5A   F3        THIS CAUSES CHIP-8 TO DISPLAY
   5B   AD        THE MEMORY PAGE WHOSE NUMBER
   5C   0D        WAS PRESSED.)
   5D   BB
   5E   D4
```

ANIMATED FACE CONTROL SYSTEM PAGE 04
TONGUE PATTERN STORAGE

```
 MA        DATA

04 00      1C 22 22 42 3C 50 00 00 00
   09      12 33 33 63 62 22 00 00 00
   12      12 33 33 63 62 23 60 60 00
   1B      12 33 33 63 62 23 20 20 00
   24      12 33 33 63 62 22 62 60 00
   2D      12 33 33 63 62 23 63 60 00
   36      12 33 33 63 62 23 23 22 00
   3F      12 33 33 63 63 22 62 62 00
   48      12 33 33 63 63 22 63 63 00
   51      12 33 33 63 63 22 62 62 00
   5A      12 33 33 63 63 22 23 23 00
   63      12 33 33 63 63 22 63 63 40
   6C      12 33 33 63 63 22 62 62 60
   75      12 33 33 63 63 22 23 23 20
   7E      12 33 33 63 63 22 63 63 60
   87      12 33 33 63 63 22 63 63 60
   90      02 33 33 63 23 22 62 62 62
   99      02 33 33 23 63 22 63 63 63
   A2      02 33 33 63 63 22 23 23 23
   AB      10 13 13 23 63 22 63 63 63
   B4      00 33 33 63 23 22 62 62 62
   BD      10 33 33 63 63 22 63 63 63
   C6      12 33 33 63 63 22 63 63 63
   CF      10 33 33 63 63 22 63 63 63
   D8      02 32 32 63 63 22 22 22 22
   E1      02 33 33 62 63 22 63 63 63
   EA      12 33 33 63 62 22 63 63 63
   F3      10 33 33 63 63 22 23 23 23
   FC      00 00 00 00
```

MORSE KEYBOARD/KEYER SYSTEM by Jeff Duntemann 9/12/77

```
MA  OPCODE

00  F8 00 B3 B4 B5 B7     Initialize high order register bytes
06  B8 B9 BA BB BC BD
0C  F8 FF A7              Initialize stack pointer
0F  F8 52 A8              Initialize DELAY PC
12  F8 74 A9              Initialize ASCII lookup table
15  F8 01 AC              Initialize Morse element counter
18  E7                    X=7
19  37 62                 Branch to KEYER if INPUT pressed
1B  D8 D8                 Call DELAY twice
1D  3E 1D                 Loop until key pressed
1F  6B                    Input ASCII byte from keyboard
20  64 27                 Output ASCII byte to display; dec. R7
22  09                    Load first value from ASCII table into D
23  32 2B                 Go to M(2B) if table byte = 0
25  07                    Load keyboard byte into D
26  E9                    X=9
27  F3                    Test for table=keyboard via XOR
28  32 34                 Go to M(34) if keyboard byte found
2A  38                    Skip next instruction
2B  1C                    Increment Morse element counter RC
2C  19                    Increment lookup table pointer
2D  8C                    Fetch Morse element counter to D
2E  FD 08                 Subtract D from 08
30  3B 00                 Go to start if Morse element counter >08
32  30 22                 Go test keyboard byte against table again
34  89                    Fetch address of found table byte to D
35  FC 2E                 Add 2E to address to point byte to encoder
37  A9                    Store encoder address in R9
38  09                    Load encoder into D
39  38                    Skip next instruction
3A  8D                    Fetch stored encoder to D
3B  FE                    Shift encoder one bit left
3C  AD                    Store shifted encoder in RD.0
3D  3B 47                 Go to M(47) if bit was 0 (dit)
3F  7B D8 D8 D8 7A D8     Form dah (Q on 3 time units, off 1 unit)
45  30 4B                 Go to M(4B)
47  7B D8 7A D8           Form dit (Q on 1 time unit, off 1 unit)
4B  2C 8C                 Decrement Morse element counter & fetch to D
4D  32 00                 Go to start if Morse character is finished
4F  30 3A                 Go to M(3A) for next element

                       DELAY
51  D0                    Return to MAIN
52  E7                    X=7
53  6C AB                 Read byte from toggles and store in RB
55  F8 FF AA              Store inner timing constant FF in RA.0
58  2A 8A                 Decrement RA & fetch it to D
5A  3A 58                 Loop again if RA ≠ 00
5C  2B 8B                 Decrement RB & fetch it to D
5E  3A 55                 Loop again if RB ≠ 00
60  30 51                 Go to RETURN
```

AN UNBUFFERED MORSE KEYBOARD PROGRAM--This program is useful in code practice, or as a good demonstration of what the 1802 can do with minimal add-ons. You do need an ASCII keyboard, but you can find them floating around used for cheap, and even new for $50 if you're careful. The VIP ASCII keyboard is very nice once you get used to the elastomer and the squeak which subs for touch feedback. (though now that I think of it, the beep might get in the way of audible Morse Code, so...maybe not the VIP keyboard) I used the Radio Shack TTL keyboard, but I don't recommend it. It uses individual TTL chips that suck power and get hot...yecechh. Fortunately a friend of mine sat on it and broke the PC board, so it's behind me now.

Whatever keyboard you use, the idea is to bring EF3 low whenever a key is pressed. Most ASCII keyboards have a Data Ready line or something like that. If your Data Ready line goes high when a key is pressed you may have to change the instruction at 00 1D to 36. When the program senses that EF3 has changed state, it will strobe a byte from an input port to the bus with a 6B INPUT instruction.

Then the fun begins. The program has what we call a "lookup table." Two lookup tables, in fact. One holds the ASCII character set, and the other an artificial "encoder byte" which casts dits as 0's and dahs as 1's. The program scans down the ASCII table, matching the keyboard byte to the bytes in the table. When it finds a match, it adds 2E to the address of the ASCII byte which matches the keyboard byte. The address now points to the encoder byte.

But...the encoder bytes for E, I, S, & H are all the same. The Morse characters for those letters are dit, didit,dididit, and didididit, respctively. So 00 will, in fact, be the encoder byte for any character made out of pure dits. The problem is, the encoder byte tells you nothing about how long a character is. That information is included in the table, in the form of 00 bytes between character groups. Hams will notice that the characters are grouped by the number of "elements' (dits or dahs) in the character. E and T (dit and dah) have one element. I, A, N, & M have two elements, and so on. A register half (RC.0) is started out with a value of one, and every time the scan software encounters a 00 in the ASCII table, it increments the element counter, RC.0. By the time the program finds the matching ASCII byte, the element counter will contain the number of elements in the character. So E, I, S, & H all come out different, as they should.

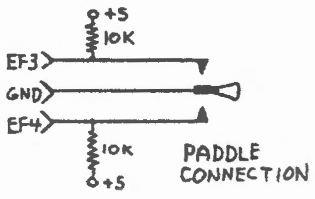
The Morse character is created by shifting bits out of the encoder byte and testing them. 0's cause a jump to a dit-former which turns Q on for a period of time. 1's cause a jump to a dah-former which turns Q on for three times as long. After each element Q remains off for one dit-length. The time period is taken from the toggle switches each time DELAY is called, so you can change the speed of the Morse characters by changing the setting in the toggle switches. I should point out that this will not work on a VIP unless you add toggle switches via an ELF-adapater device like I described previously.

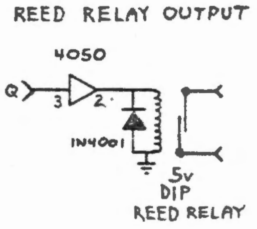
A bonus in this program is a little electronic keyer routine which is entered by depressing INPUT at the time you start the program running. The keyer is used by connecting paddles to EF3 and EF4 and grounding them to produce a continuous string of dits and dahs. The dits and dahs are self completing, but the program has no memory, so you must let the current dit or dah finish before jumping ahead to the next one.

This problem extends to the keyboard program, where it is more serious. Generally, people can type more quickly than they can copy Morse code, so it would be very easy to get "ahead" of the program. Unfortunately, while the program is generating characters it ignores the keyboard. You must wait until one character is completed before typing in the next letter. I am in the process of writing an interrupt-driven buffered keyboard program, in which the keyboard causes an interrupt which stores the input ASCII byte in a 256-byte FIFO (First In First Out) buffer. With such a program you could type merrily away and get 256 letters ahead of the computer before you start eating your own tail and losing data. Sadly, at press time it doesn't work yet. Maybe one of these issues...

```
MA  OPCODE
                              KEYER
62  36 6C                         Test EF3 for dah select
64  3F 62                         Test EF4 for dit select
66  7B D8 7A D8                   Form dit
6A  30 62                         Go back to KEYER start
6C  7B D8 D8 D8 7A D8             Form dah
72  30 62                         Go back to KEYER start

                              LOOKUP TABLES

74  45     ASCII E                 A2  00      E Encoder
75  54           T                 A3  80      T
76  00                             A4  00
77  49           I                 A5  00      I
78  41           A                 A6  40      A
79  4E           N                 A7  80      N
7A  4D           M                 A8  C0      M
7B  00                             A9  00
7C  53           S                 AA  00      S
7D  55           U                 AB  20      U
7E  57           W                 AC  60      W
7F  4F           O                 AD  E0      O
80  47           G                 AE  C0      G
81  44           D                 AF  80      D
82  4B           K                 B0  A0      K
83  52           R                 B1  40      R
84  00                             B2  00
85  48           H                 B3  00      H
86  56           V                 B4  10      V
87  46           F                 B5  20      F
88  42           B                 B6  80      B
89  4A           J                 B7  70      J
8A  43           C                 B8  A0      C
8B  51           Q                 B9  D0      Q
8C  4C           L                 BA  40      L
8D  58           X                 BB  90      X
8E  50           P                 BC  60      P
8F  59           Y                 BD  B0      Y
90  5A           Z                 BE  C0      Z
91  00                             BF  00
92  31           1                 C0  78      1
93  32           2                 C1  38      2
94  33           3                 C2  18      3
95  34           4                 C3  08      4
96  35           5                 C4  00      5
97  36           6                 C5  80      6
98  37           7                 C6  C0      7
99  38           8                 C7  E0      8
9A  39           9                 C8  F0      9
9B  30           Ø                 C9  F8      Ø
9C  2F           /                 CA  90      /
9D  2D           -                 CB  88      -
9E  00                             CC  00
9F  3F           ?                 CD  30      ?
A0  2C           ,                 CE  CC      ,
A1  2E           .                 CE  54      .
```

The best way to test this program is to use Q to drive a reed relay, and then drive a CPO (Code Practice Oscillator) with the reed relay. That way you could eventually use the reed relay to key your transmitter. Do take care to keep stray RF out of the computer. The first time I tried it, a raging tide of RF leakage raced through my power supply and wiped all 2K of RAM clean as a whistle! Dahdidahdit dahdahdidah!!





VIDEO SCRATCHPAD PROGRAM   BY JEFF DUNTEMANN   REV 9   2/3/78

```
MA  OPCODE

00  F8 00 B1 B2 B3 B5   Initialize high-order registers
06  B6 B8 A9 AB         and byte pointers
0A  F8 01 B9 BB BD
0F  F8 80 AC AA         Initialize bit pointers
13  F8 28 A1            Initialize interrupt PC
16  F8 FF A2            Initialize stack pointer
19  F8 66 A3            Initialize MAIN PC
1C  F8 5A A5            Initialize BNKWRT PC
1F  F8 4F A6            Initialize DOTWRT PC
22  F8 45 A8            Initialize DELAY PC
25  D3                  Begin executing MAIN PC
                    INTERRUPT
26  72 70               Restore D, X, & P
28  22 78 22 52         Push P, X, & D onto stack
2C  C4 C4 C4            NOP's for sync delay
2F  F8 01 B0 F8 00 A0   Re-point R0 to display page
35  80 E2               Prepare for first DMA cycle
37  E2 20 A0            DMA reset
3A  E2 20 A0            DMA reset
3D  E2 20 A0            DMA reset
40  3C 35               Test for refresh done
42  30 26               Go to return
                    DELAY
44  D3                  Return to MAIN
45  F8 07 BE            Load timing constant into RE
48  2E                  Decrement RE
49  9E                  Load RE.1 into accumulator
4A  3A 48               Loop again if not done
4C  30 44               Go to return
                    DOTWRT
4E  D3                  Return to MAIN
4F  89 AB               Update byte pointer
51  8A AC               Update bit pointer
53  EB                  X=B
54  F1                  Combine bit pointer & screen via OR
55  5B E2               Write dot to screen
57  30 4E               Go to return
                    BNKWRT
59  D3                  Return to MAIN
5A  89 AB               Update byte pointer
5C  8A AC               Update bit pointer
5E  FB FF               Inverts D via XOR IMMEDIATE
60  EB                  X=B
61  F2                  Combine bit pointer & screen via AND
62  5B E2               Write blank to screen
64  30 59               Go to return
```

```
                        MAIN
66  E2 69                   Turn CDP 1861 on
68  3F 75                   Skip clearing routine unless INPUT pressed
6A  F8 FF AD                Point RD to top of display page
6D  ED                      X=D
6E  F8 00 73                Store 00 on screen & decrement pointer
71  8D                      Load pointer into D
72  3A 6E                   Loop again if not done
74  5D                      Store 00 in last byte of display page
75  E2 6C                   Input toggles
77  F6 33 89                Tests "move right" bit & branches
7A  F6 33 98                Tests "move left" bit & branches
7D  F6 33 A7                Tests "move down" bit & branches
80  F6 33 AF                Tests "move up" bit & branches
83  F6 3B B7                Tests dot/blank bit
86  7B                      Turn Q on
87  30 B7                   Go to EXECUTE
89  BA                      Store D in RA.1
8A  8A                      Fetch temporary bit pointer
8B  F6 33 92                Shift right and test for border cross
8E  AA                      Update bit pointer
8F  9A 30 7A                Put old D back in D & return to shift & test
92  19                      Increment temporary byte pointer
93  76                      Shift bit back into other end of bit pointer
94  AA                      Update bit pointer
95  9A 30 7A                Put old D back in D & return to shift & test
98  BA                      Store D in RA.1
99  8A                      Fetch temporary bit pointer
9A  FE 33 A1                Shift left and test for border cross
9D  AA                      Update bit pointer
9E  9A 30 7D                Put old D back in D & return to shift & test
A1  29                      Decrement temporary byte pointer
A2  7E                      Shift bit back into other end of bit pointer
A3  AA                      Update bit pointer
A4  9A 30 7D                Put old D back in D & return to shift & test
A7  BA                      Store D in RA.1
A8  89                      Fetch temporary byte pointer
A9  FC 08                   Add 08 to D & put sum in D
AB  A9                      Update byte pointer
AC  9A 30 80                Put old D back in D & return to shift & test
AF  BA                      Store D in RA.1
B0  89                      Fetch temporary byte pointer
B1  FF 08                   Subtract 08 from D & put difference in D
B3  A9                      Update byte pointer
B4  9A 30 83                Put old D back in D & return to shift & test
```

```
                        EXECUTE
B7  D5 D8 D6 D8             Generate one "wink" of cursor
BB  31 C0                   Go to M(C0) if Q is on
BD  D6                      Call DOTWRT & write on screen
BE  30 68                   Go to test for clear
C0  D5                      Call BNKWRT & write on screen
C1  7A                      Turn Q off
C2  30 68                   Go to test for clear
```

HOW TO USE THIS PROGRAM: Load it carefully through the toggle switches. No peripherals but the Pixie video chip and monitor are required. All control is through the toggles and the INPUT switch. When the program is run, memory page 01 is displayed on the screen. Normally, just after power-up, this page will contain random bits scattered throughout, especially if it is CMOS RAM. To begin with a clear (all black) screen, hold INPUT depressed while flipping the RUN switch up. The routine at M(6A-70) will store 00 in all memory locations within page 01, blanking the screen. The screen can be cleared at any time by pushing INPUT.

When run, a winking cursor will appear in the extreme upper left hand corner of the screen, corresponding to bit 7 of M(01 00). By flipping the first four toggle switches up and down, the cursor will move around the screen, leaving a "trail" of white behind it. If the fifth toggle is flipped up, the cursor will "eat" white areas on the screen by writing 0 bits behind it. The cursor appearance is always the same, regardless of its "polarity." To remove the cursor from the middle of a complicated design without disturbing the screen pattern, reset the program by flipping RUN down and up again. The cursor will be sent back to its home position. The screen pattern will not change.

The clearing routine can be modified to fill the screen with a white background to draw upon in black by changing the byte at M(6F) to FF. Toggle 0 makes the cursor move to the right. Toggle 1 makes it move to the left. Toggle 2 makes it move downward. Toggle 3 makes it move upward. If the cursor is sent off any side of the screen, it will appear immediately on the opposite side. Kaleidoscopic displays can be created by greatly increasing the speed of the program. The delay subroutine should be re-written thus:

```
D3 F8 01 AE 2E 8E 3A 48 30 44
```

The cursor will then move at great speed, and by randomly flipping the direction-control toggles a crazy-quilt pattern will continually weave on the screen.

Enjoy it!

YE OLDE INDEXE;

A CRITICAL LOOK AT THE COSMAC VIP 7
AN ANIMATED VIDEO FACE SYSTEM 18
AN ELF ADAPTER BOARD FOR THE VIP 10
AN EPROM CARD FOR THE VIP 8
AN UNBUFFERED MORSE KEYBOARD PROGRAM 22
BINARY ARITHMETIC SUBROUTINE BOOK 14
CD vs. D version 1802's 3
CMOS RAM BARGAIN 1
COSMAC I/O PORTS 2
COSMAC PRICE BREAKTHROUGH 7
COSMAC'S FATAL FLAW 7
COSMO'S FACE 9
Disabling ELF RAM 12
Disabling interrupts 17
Eastman 910 warning 15
ELF address display circuit 7
ELF/VIP software compatibility 10
ELF-II expansion bus pinout 12
EPILOGUE 26
EVERYTHING YOU NEVER WANTED TO KNOW ABOUT PAPER TAPE 3
FAST/SLOW CLOCK GENERATOR 15
GIVE YOURSELF ROOM 2
"Good King Wenceslaus" Super Sound listing 13
H-10 paper tape reader/punch 3
HAMFESTS 1
Hex keyboard circuit 4
HOW FAST RAM? 6
HOW MUCH RAM? 6
I/O strobe expansion using 4028 3
IN SEARCH OF COSMAC STANDARDS 11
Interrupt enable flipflip (IE) 16
INTERRUPTS MADE SIMPLE 16
IO-BUS 11
IT MAY OR MAY NOT COME IN THE MAIL 1
Loading large programs 14
Lookup tables 22
Mail order reviews 1
Morse keyboard/keyer program 21
OP-80A bootstrap loader program 3
OP-80A paper tape reader 3
Photo of Cosmo the Robot 9
PIN-8 14
Plastic "consumer" version 1800 chips 7
POWER SUPPLY POINTERS 2
Program debugging with memory address displays 6
Programmers' paranoia 14
RAM CHOICES 2
RAM speed vs. clock speed table 6
Recomp-1 7
Reed relay driver from Q 22
Relocatable ROM block 8
Robot face displays 19
Robot face program 18
Sample interrupt routine 17
Source code vs. object code 14
Stack 16
State code lines (SC0 & SC1) 7
STUDIO II CONVERSION 7
Super Sound block diagram 13
SUPER SOUND REDUX 13
TEHOPS (Tape Entered Hex Op System) 4
TETOPS (Tape Entered Toggle Op System) 5
THE SOURCE WILL ALWAYS BE WITH YOU 14
THE 1802 MASS STORAGE PROBLEM 5
TickTockTech ad 8
Toggle switch warning 2
UNDER NO CIRCUMSTANCES 1
Vanishing bootstrap paper tape loader program 6
VERSION, VOLTAGE, & SPEED 3
Video scratchpad program 23
VIP expansion bus pinouts & description 11
VIP I/O line summary 11
WATCH OUT 2
VIP's I/O ports 12
WHEREZAT PROGRAM AT? 6
Wire wrapping 1
WIRING WOES 1
X-BUS 11
ZOUNDS! 10
Zounds program 10
1802 speed limitations 7
1804 & 1806 rumors 26
1852 I/O port 2
1863 13
2114 RAM circuit & pinout 2
2758 EPROM circuit & pinout 8
4515 decoder/latch pinout 4
4556 decoder pinout 8

EPILOGUE--Look, don't be fooled. I've tried my damndest to make the 1802 sound attractive, and to make messing around with computers on a fundamental level sound irresistable. I've cautiously omitted tales of torn hair involving bad solder joints on connector cables, leaky bypass caps, and program glitches that must be due to a bad CPU chip. (That hasn't happened yet. I've been waiting for three years...) I've omitted accounts of dodging Unidentified Flying Pans for spending too much time crouched over a piece of perfboard crawling with bugs. If you have Elves in your house, all of this should be familiar. If you do not yet have an ELF--well, the entire ELF series is available in reprint from Popular Electronics Magazine; write to them for particulars. You really oughta build one. Really.

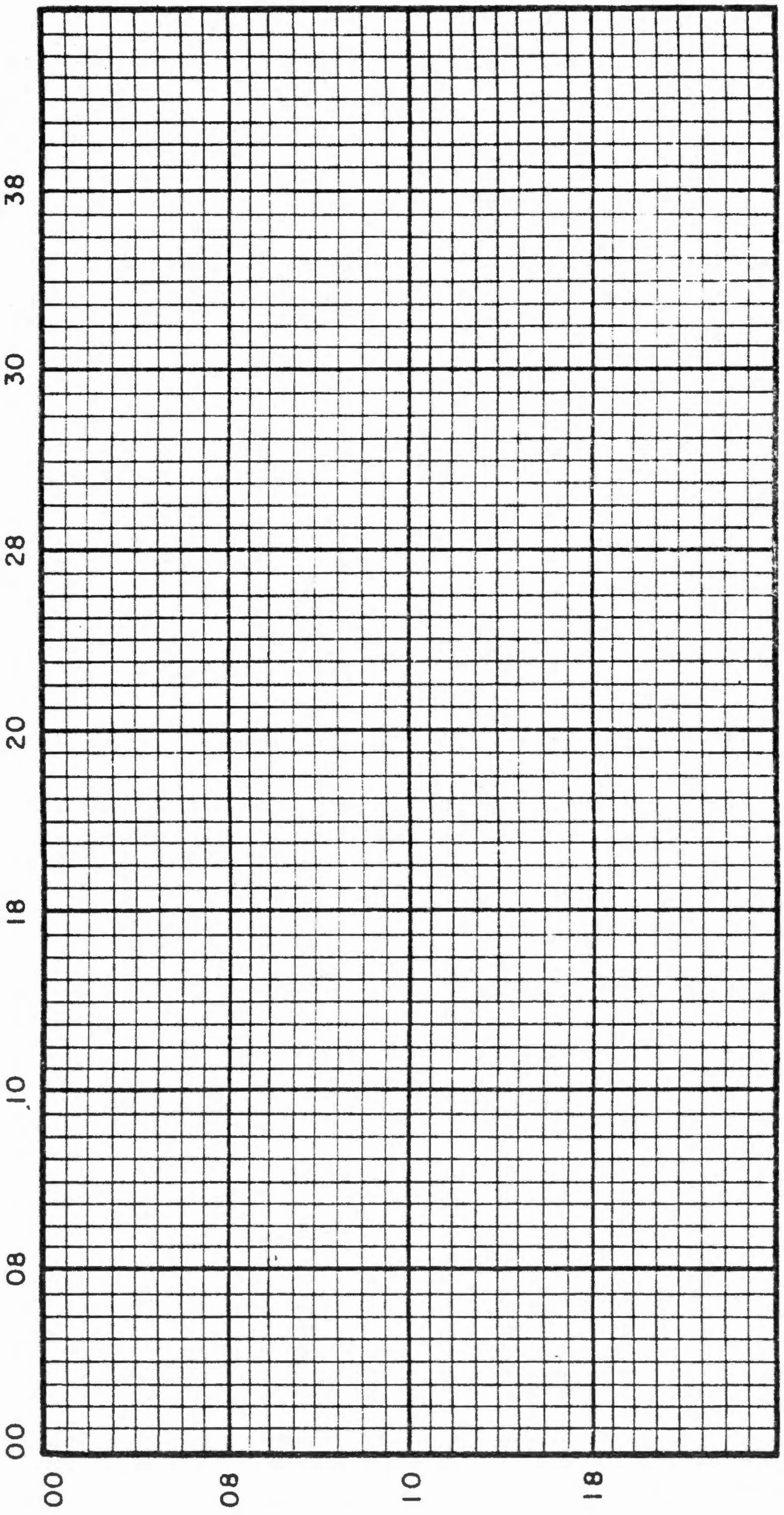
I built my first ELF in 1976, and started writing this book in 1977. Here it is, the first month of 1980. Where do we go from here? Well, there's the tale of the CDP1804. Chrysler Corporation asked RCA to design a single-chip microcomputer to handle their new emission control system. RCA saw dollar signs, sat down, and came up with the 1804. On one chip are 2K ROM, 64 bytes RAM, and a programmable counter/timer that can measure the length of pulses, generate square waves, count events, and generate processor interrupts. Best of all, the 1804 has 33 new instructions, all two-byte opcodes beginning with 68. Now you know what RCA was saving that mysterious opcode 68 for. There is a single instruction to load both halves of a register with immediate data, or data from R(X). You can now copy an entire register to R(X) with a single instruction. You can disable or enable interrupts by setting IE directly with a single instruction that affects nothing else. All housekeeping for Standard Call & Return are now handled by two instructions. Beautiful. Now then, the only snag is that Chrysler is about to go broke, leaving RCA with ten million 1804's, all mask programmed to count smog particles coming out of exhaust manifolds. Even if Chrysler survives, RCA will have its hands so full making Chrysler's 1804's that it won't be putting the chip into general release. The only way you'll be able to get one is by taking your Dodge Aspen apart. So it goes.

A marginally reliable source says that Hughes Corporation is designing the CDP1806, which will be upwardly compatible with both the 1802 and the 1804. Since there are now 223 unused '68' opcodes, that leaves them a lot of room. No bets on what the 80's will bring. An 1816? A CMOS 2764? No bets, guys. No bets at all.

If anybody wonders what you do with your evenings, just tell them what I do: Come on over and meet my computer some day.

It's the one with the pointed ears.

CHIP 8 DISPLAY



Reprinted from the
RCA *Engineer*

CHIP 8 - Program Sheet

Page

Date

Program Name:

Address	Code	Comments
_ _ _ 0		
2		
4		
6		
8		
A		
C		
E		
0		
2		
4		
6		
8		
A		
C		
E		
0		
2		
4		
6		
8		
A		
C		
E		
0		
2		
4		
6		
8		
A		
C		
E		

Reprinted from the RCA
Engineer

THE 1802 SOURCE !

Ne'r was there a well so deep...

WHEN your microprocessor is #5, you don't just try harder, you have to dig halfway down to the Moho for parts, hardware, and software. I've done some heavy digging in the last three years, so it might be neighborly to lay out what I've found so it can be useful. This is the best list of COSMAC suppliers I've been able to put together. It's in no particular order. Use it in good health!

NETRONICS R&D LIMITED; 333 Litchfield Rd. New Milford CT 06676; Phone: (203) 354-9375. This is the home of the ELF-II, an excellent kit-form 1802 machine. Lots of hardware and software add-ons available, including ASCII keyboard, color, music synthesis, BASIC, light pen, editor/assembler/ disassembler, and lots more coming up. Famous for using smaller print in their ads than I use in this book, but that's cool.

QUEST ELECTRONICS; PO Box 4430C, Santa Clara CA 95054; Phone: (408) 988-1640. Quest is a super-fine mail-order outfit for parts in general. Their Super ELF kit is as good as or better than the ELF-II. It has a ROM monitor and richer I/O, tho not as much software at this time. Quest also publishes Questdata, a monthly software newsletter for 1802 users, for $12/year. Until very recently Quest also advertised a basic ELF kit for about $90. This may no longer be available. The more I see of the Super ELF, the better I like it. Get their catalog, and Questdata.

ANACRONA; PO Box 2208P, Culver City CA 90230; Phone: (213) 641-4064. Anacrona now carries a full line of the plastic (LE) series of 1802 devices. They're <u>cheap</u>, people. The 1802 is $9.90; the 1852 $1.90; the 1854 UART $8.50. For my money the plastic case is <u>better</u> than the ceramic; the legs don't fall off as easily. This is the place for individual chips.

H. C. WILL SOFTWARE; PO Box 347, Pinebrook NJ 07058. This is a new one, and I have no experience with him, but he sells VIP and possibly ELF software. Send an SASE for his program list. Can't hurt to try.

ARESCO; 6303 Golden Hook, Columbia MD 21044. ARESCO won't say what its damned acronymic name means, but it's a tremendous outfit that publishes the VIPER for VIP people and the STUDIO TUTOR for owners of converted Studio II TV games. VIPER is something no 1802 owner should miss; there is plenty of data applicable to all systems, not just VIPs. VIPER costs $15/year and worth it at twice the price. ARESCO also sells a number of good 1802 programs, a series of VIP books called "Pips for VIPs" and newsletters for the Sorcerer and PET. In addition, they also retail RCA add-on hardware boards and a Studio II-ELF conversion kit. They must be doing it for love; I doubt they're making much money on any of their ventures. Four stars and a gold-plated 2716!

THE LITTLE LEPRECHAUN NEWS; PO Box 57784, Webster TX 77598; Published by Charles E Manry. I haven't seen this, and only heard about it recently. You might send an SASE and a few extra stamps for a sample issue. Anything with such a cute title <u>has</u> to be either very very good or a severe emetic.

COSMAC USER'S GROUP; PO Box 7162, LA CA 90022. Watch out for this one; I've received only three issues spaced out over a year and a half, and nothing since May '79. The publisher, Patrick Kelly, sounds flakier than even me, and that's going some. He blamed the long delay between two of the issues on an amnesia attack following shock treatment for severe depression. Yeah, I sympathise, but what the hell--it's a passable newsletter when it shows up. Caveat Elfhackere.

DIGITAL SERVICE & DESIGN; PO Box 741, Newark OH 43055. Wyndham Davies is in charge. Unless he's gone bankrupt or gotten disgusted with us, DS&D makes the DSD-1802 computer. It sounds like a helluva system, including 16K RAM card, 32K EPROM card, I/O network card (whatever that is) and floppy disk controller card (!). A whole set of boards can be had for $125. I know nothing more about him at this time. Again, it never hurts to write and ask. Tell him you saw it in Captain Cosmo's Whizbang.

CUDDLY SOFTWARE; 98 Thorndale Terrace, Rochester NY 14611; Phone: (716) 328-8259. No, this is not me, though he's scarcely three miles up the road. What he sells, at this time, are two mighty fine programs, an OP System and a Trace Program. The Op System is a collection of subroutines that formats screen resolution and moves bytes around. I can't do it justice here; you should write for the specs. The Trace Program is the intriguing one, allowing single-stepping of a program with dynamic display of processor registers after each step! It requires an ASCII keyboard and 3K minimum, but it sounds damned useful. No price info at this time. Check it out!

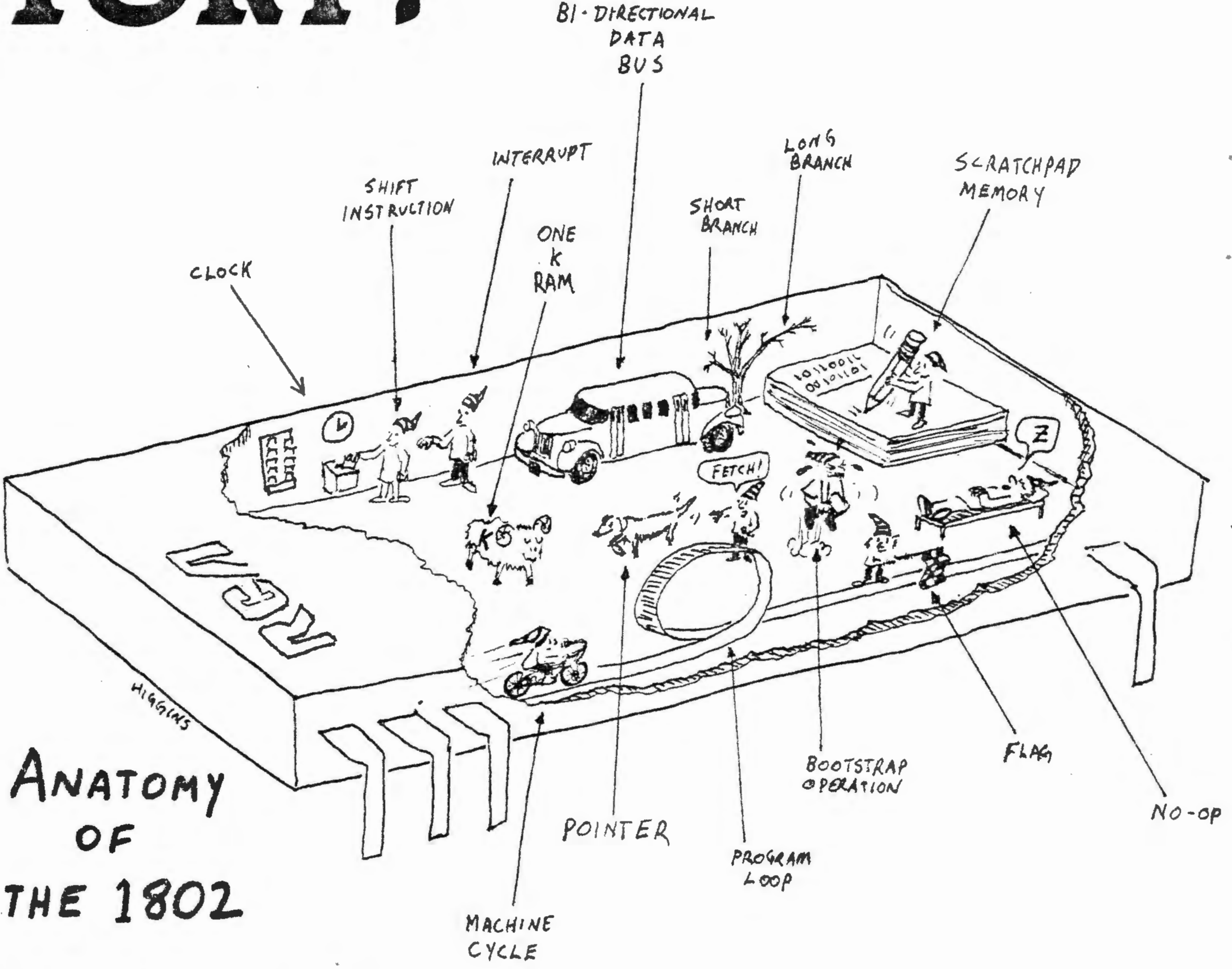
O. C. STAFFORD ELECTRONICS; 427 S. Benbow Road Greensboro NC 27401. Ozzie Stafford is a ham and a great guy to deal with. He sells an 1802 computer board oriented toward ham radio and repeater control, plus lots of other circuit boards relating both to computers and ham radio. Everything of his I've seen has been terrific. Send for a list.

MICRO-SYSTEMS; 807 Cedar Circle, Spencerport NY 14559. Lee Hart, proprietor. Lee sells what is undoubtedly the most sophisticated piece of 1802 engineering I've ever seen: The SBC100-X1 single board 1802 computer. On a tiny little 4 by 6 inch card he's got 2K RAM, up to 4K ROM, 2 latched parallel ports, 2 optically isolated serial ports for RS232 or 20 mil current loop, bus terminator resistors, logic for 80 multiplexed I/O lines, and miscellaneous whatnot. It requires some offboard I/O to talk to it, but it can be jumper-configured almost any way you want it. An excellent monitor, IDIOT/4, is available on ROM, and Lee will soon make FORTH available. This board is <u>not</u> for the beginner in the same way the ELF is, but if you've made an ELF work you can handle it. Utterly, utterly first class!

RCA COSMAC VIP MARKETING; New Holland Avenue, Lancastewr PA 17604. Yes, kids, you can in fact order VIP products direct from RCA. I did it, and it took 2 whole months for the stuff to get in. Not an enviable track record, but it can be done. RCA sells Super Sound, Simple Sound, ASCII Keyboard (nice, by the way, and cheap) Color, Expansion Board, and some other odds and ends. Send for the VIP list for current prices. RCA also sells a COSMAC evaluation kit for about 400 bucks, which is a big board with a TTY interface and room for 4K RAM. I don't think it's worth the money, but suit yourself. The Microtutor II is an ELF but not as nice, and a <u>whole</u> lot more expensive. RCA also sells big mainframes using the 1802 in the 5-10 kilobuck range. If you're that rich you shouldn't be reading trash like this anyway. On the whole, except for the VIP expansion boards, you casn do much better elsewhere.

OPTIMAL TECHNOLOGY; Blue Wood 127, Earlyville VA 22936; Phone: (804) 973-5482. They sell several EPROM programmers with software for the 1802. I've heard nothing more about the devices though, and have been itching to order one for some time. Write for specs and a catalog is about all I can say.

THE INSIDE STORY:



ANATOMY OF THE 1802

Jeff Duntemann
301 Susquehanna
Rochester NY 14618

THE RETURN OF THE LAST WHOLE WHIZBANG CATALOG!

(Access to COSMAC socket wrenches...)

Hi, yawll::

After skulking around in the underground for more than a year, Captain Cosmo's Whizbang has finally made the big time with a Real Book Review in Kilobaud courtesy satisfied reader Larry Stone. Orders are coming in once again, long after I had pretty much given up hope of selling another copy. Many of you have written asking the price, well, the price was on the review when Larry sold it, but Wayne Green (who seems not to believe in Free Plugs for kitchen table operations) corflued it out. Thank you all for writing. Let's see what I can tell you.

CCW is still available. $5.00 plus .75 postage. Handling free, as is the grape jelly on the envelopes resulting from the handling. I mean, this is a real Kitchen Table Operation. Although I don't take credit cards, checks are cool, as is cash, for that matter. My mailroom staff is honest, since my mailroom staff is me, and I long ago learned not to cheat at solitaire. In fact, since there's no such thing as a $5.75 bill, if you send a five-spot the postage is on the house.

For those of you who haven't run across a Whizbang yet, it's probably unlike any other publication in the hobbyist world. Like all the rest of us, I started out in microcomputers as a total ignoramus, with the difference that I was an ignoramus with a long memory. I remembered what it was like to be stuck up a familiar creek with no stick and nobody to run to for help, and I hoped to be able to ease a few of you over some of the hard spots.

Parts of CCW are over three years old now, and the micro world has aged some. Would you believe I paid $39.95 for my first 1802? (Still have the little bugger, too.) Today, a good scrounger with a wire-wrap gun can make an unbelievably cheap computer. 2114 RAM is now three or four bucks a shot--cheaper than a first run movie. There is still no better, cheaper way to get into computers than via the 1802. Order the Popular Electronics 'Elf' reprint series first, and CCW second, and you'll be on your way.

What am I working on these days? PASCAL, mostly. That, and the 8086 on the S100 bus, and other various and sundry. Why did I abandon the 1802? Money. I make more money on a single computer-magazine article than I did on the whole CCW project. That, and somebody else gets to worry about mailing things out and screwups at the printers'. On the other hand, nobody else would have allowed me to do CCW in their magazine as I wanted to do it--so for that alone, it was worth the effort.

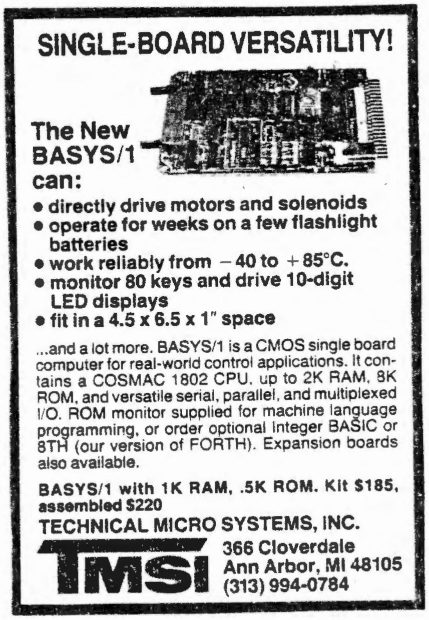
What new 1802 gossip? That's the problem--there isn't any. RCA seems to be abandoning its problem child, and the 1804 seems to have sunk like the Titanic. People are flocking to National Semi's CMOS

Z80 lookalike. The future of the 1802 looks kinda grim from here.

But there is yet one bright spot in the 1802 scene: TMSI. This is the same outfit I listed in CCW as Micro Systems, but they changed their name and moved to Michigan not long after CCW went to press. Since then their single board 1802 computer has gone through several revisions and is now the BASYS/1, and is truly remarkable. The future of the 1802 is with threaded code languages, like FORTH and TMSI's similar 8TH. Hang BASIC--that's for kids. Get yourself a BASYS and try 8TH--it is truly the ONLY 1802 machine I unhesitatingly recommend.

So--see what you can do with the 1802. As always, you're pretty much on your own. The magic you create will be of your own making--but that's the very best kind of magic there is. Best to you all--it's been fun. Read the computer mags and you'll bump into me now and then, but to be honest, you'll never hear me say things quite the same way I said them in CCW.

Cheers, and keep on hackin'--



Jeff Duntemann

Jeff Duntemann KB2JN

Carol and Jeff Duntemann
301 Susquehanna Rd.
Rochester, NY 14618