VIPER

June - July 1982

Volume 4, Number 2 Journal of the VIP Hobby Computer Assn.

The VIPER was founded by ARESCO, Inc., in June 1978

****

Contents

VIPHCA INFO...

The VIPER, founded by ARESCO, Inc. in June 1978, is the Official Journal of the VIP Hobby Computer Association. Acknowledgement and appreciation is extended to ARESCO for permission to use the VIPER name. The Association is composed of people interested in the VIP and Computers using the 1802 microprocessor. The Association was founded by Raymond C. Sills and created by a constitution, with by-laws to govern the operation of the Associtionn. Mr. Sills is serving as director of the Association, aw well as editor and publisher of the VIPER.

VIP and COSMAC are registered trademarks of RCA Corp. The VIP Hobby Computer Association is in no way associated with RCA, and RCA is not responsible for the contents of this newsletter. Please send all inquiries relating to the VIPER to VIPHCA, 32 Ainsworth Avenue, East Brunswick, NJ 08816.

The VIPER will be published six times per year and sent to all members in good standing. Issues of the VIPER will not carry over from one volume to another. Annual dues to the Association, which includes six issues of the VIPER, is $12 per year. Membership in the VIP Hobby Computer Association is open to all people who desire to promote and enjoy the VIP and other 1802 based systems. Send a check for $12 in U.S. funds payable to "VIP Hobby Computer Assn." c/o Raymond Sills, 32 Ainsworth Avenue, East Brunswick, NJ 08816. People outside the U.S., Canada and Mexico please send $18, due to additional postage charges. The VIPER is normally sent via first class mail, and airmail to members outside North America.

Contributions by members or interested people are welcome at any time. Material submitted by you is assumed to be free of copyright restrictions, and will be considered for publication in the VIPER. An honorarium payment is made to those whose material is published in VIPER to help cover the cost of a submission. Articles, letters, programs, etc., in camera-ready from on 21.5 x 28 cm (8.5 x 11 inch) paper will be given preferential consideration. Please send enough information about any program so that readers can operate the program properly. Fully documented programs are best, but memory dumps are okay if you provide enough information to run the program.

If you write to VIPER/VIPHCA, please indicate that it is okay to print your address in letters to the editor, if you want your address revealed to VIPER readers. Otherwise, we will not print your address in VIPER.

ADVERTISING RATES.....

1. Non-commercial classified ads from members: 5 cents per word, minimum of $1.
2. Commercial ads and ads from non-members: 10 cents per word, minimum of $2.
3. Display ads from camera ready copy: $6/half page, $10/full page.

Payment must accompany all ads. Rates subject to change.

EDITORIAL

One of the suggestions which has come up from time to time is to establish a library of VIPER's programs on tape cassettes. The purpose of the library would be to enable interested members to obtain the programs published in VIPER without having to "dink-in" all the code by hand. Hand entry of programs always opens the possibility of introducing errors (above and beyond those caused by typos) and almost always takes a lot of time. So, if tape cassettes of the various programs were available, a great deal of time would be saved and perhaps a lot of frustration eliminated. Of course, problems can also occur with tape, as most of us know. Phase reversal and head alignment variations almost always cause trouble. I've often had problems with tapes recorded at low level so that the desired signal gets muddied up by tape noise.

But still, the idea of having a program that you're interested in available on tape is very appealing. Of course, there is no "free lunch" with this idea. It would mean having to set up the library, cataloging the available material, and sending it to those who request it. Since VIPER is running on a non-profit basis, we could probably offer tape programs at a very modest charge to those who want them. Blank tape costs between 50 cents and a dollar, postage and handling around 50 cents. If we allow 50 cents to cover the amortized cost of a tape machine or two, we should be able to offer cassettes at a cost of $2.00 to $2.50. And each cassette could contain several programs, say, all the programs in a particular VIPER.

Let me know what you think of this idea. I think that the price is OK, especially compared to many of the commercial products for other computers. Of course, this idea would also depend heavily on the co-operation of VIPER authors, and we would not offer programs which are under copyright without permission of the author. And some authors may wish to reserve the right to make their programs commercially available, either on their own or by way of a software publisher, even though they might send in the program for publication in VIPER.

By the way, I'd like to call to the attention of those of you who read BYTE magazine the very fine article in the July '82 issue by Art Makosinski, "Tuning Up the 1802." It's on page 442. Art uses the VIP as a Music Composition Trainer with the VP-595 tone generator board and an ASCII keyboard to play tones and display the sequence of notes on the monitor screen. Nifty! Another victory for CHIP-8.

GIVE YOUR VIP A VOICE

by Bill Fisher *

When Netronics R & D Ltd. announced their Electric Mouth, it seemed to me that the Elf II version of that board should be compatible with the VIP, since both the VIP and the Elf II use the same (1802) microprocessor. As it turned out, the procedure for interfacing the Elf II EM board to the VIP was fairly straight forward. I thought other VIP'ers might be interested in what I did to get my VIP to talk. Incidently, the quality of the speech produced by the EM is excellent and uses the Digitalker set of chips made by National Semiconductor.

The parts required(besides the EM board itself) are a mating 86 pin connector (available from Netronics), a 44 pin board to mate with the J1 Expansion Interface connector on the VIP (Vector 3662 or similar) and two 22k 1/4 watt resistors. The 44 pin board is cut to a length of approximately two inches and the 86 pin connector attached to it with epoxy or other glue. This assembly forms an adapter which permits plugging the EM into J1 of the VIP. Table I lists the required wiring cross-connections to be made on the adapter.

On the EM itself, disconnect the "wait" jumper and reinstall it at the EF4 location. In order to provide for powering the EM with 5 volts directly from the VIP, short the input to the output of Q1 (7805) by providing a jumper between the two outside terminals of Q1. (I measured the current drawn by the EM to be 180 mils which the original VIP power supply module seems to handle very nicely.) Also disconnect and tape one end of R10 (4.7k) from the EM board. The pull-up to 5 volts is already provided by a 22k resistor in the VIP.

The schematic of the changes to the VIP is shown in Figure 1. Pins 9 and 10 of J1 are made available for use as I64 and I66 respectively, by making the cuts on the top of the VIP board as indicated in Figure 2. The I66 signal must be buffered to permit it to drive a TTL input on the EM board. Instead of adding an additional IC for this, I chose to use the spare gate available on the VIP in U26. On the later VIP's, which came assembled, U26 is soldered directly to the board. In this case, cutting pin 9 close to the board and bending it up is the simplest method of gaining access to pin 9. If you have an earlier VIP, U26 may have been installed with a socket, in which case pin 9 can be easily lifted from the socket. I have done this modification successfully to both versions of the VIP. The wiring and resistor placement on the underside of the VIP are shown in Figure 3.

I originally used the existing I63 and I61 in place of I64 and I66 respectively. This can be done if you wish, and simplifies the modification somewhat inasmuch as you can merely connect the leads from pin 9 and 10 of J1 to the appropriate ends of existing resistors R19 and R36. I decided, however, to make I64 and I66 available as described above to eliminate any conflict with other existing uses of I61 and I63.

* 2 Barnard Road, Armonk, N.Y. 10504

I have included two programs so that you may immediately verify the operation of your EM board. The first one will output the entire EM vocabulary. The second program is the familiar high-low number guessing game. The EM will first announce that it has a number from 1 to 99 and as your guesses are inputted via the hex keyboard, the EM will repeat the number and let you know if it is too low, too high or correct.

I will be glad to correspond with anyone regarding this modification if they include a SASE. For those interested, I also have a program which permits you to conveniently experiment with creating new words from bits of the available EM vocabulary.

TABLE I

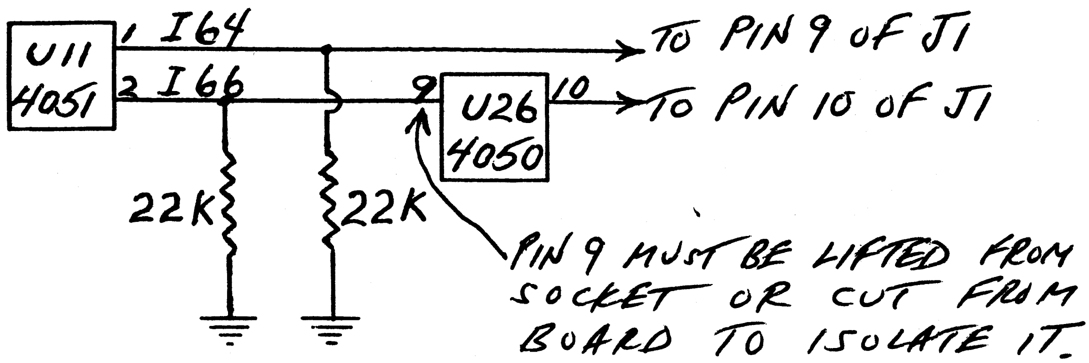
EM PINS	VIP J1 PINS	SIGNAL
5 & 7	Z & 22	GND
9 & 11	Y & 21	+5 V
14	18	$\overline{\text{RUN}}$
30	W	$\overline{\text{MRD}}$
32	11	TPB
34	V	BUS 7
38	U	BUS 6
42	T	BUS 5
46	S	BUS 4
50	R	BUS 3
53	10	I66 (formerly $\overline{\text{SYNC}}$)
54	P	BUS 2
57	9	I64 (formerly SPOT)
58	N	BUS 1
62	M	$\underline{\text{BUS}}$ 0
78	2	$\overline{\text{EF4}}$

ELECTRIC MOUTH VOCABULARY LIST PROGRAM

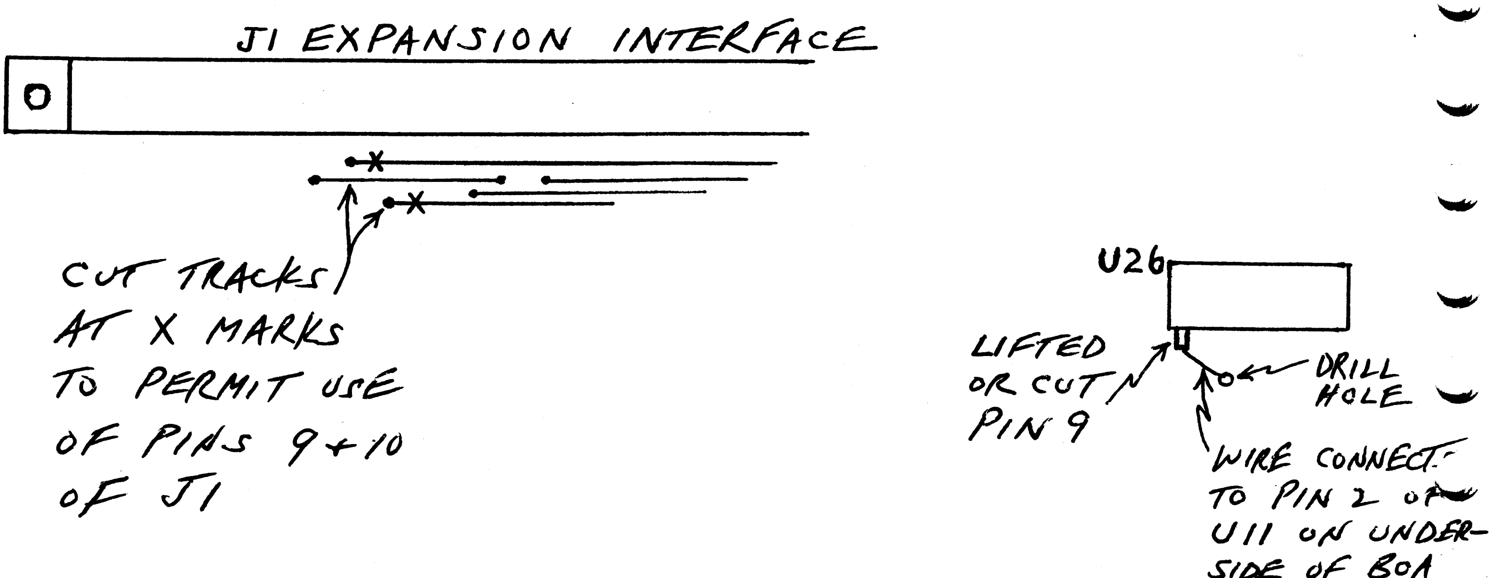
```
0000   F8 00 B3 B4 F8 0F A3 F8 12 A4 F8 01 54 E3 D3 64

0010   FC 66 XX 37 13 66 47 37 17 04 FC 01 54 FB 90 32

0020   23 30 11 F8 00 B0 A0 E0 D0 00 00 00 00 00 00 00
```

(XX at M(0012) supplied by program)

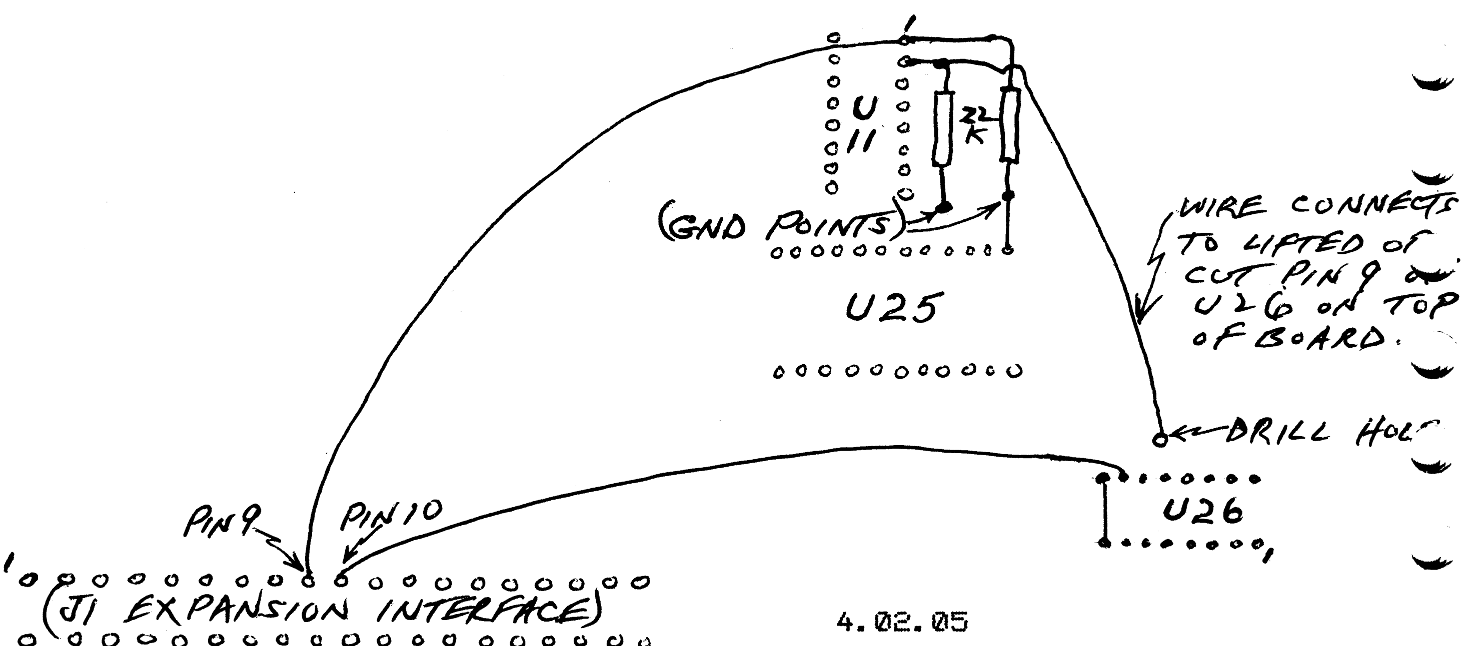
SCHEMATIC - FIG 1



TOP VIEW OF VIP - FIG 2



BOTTOM VIEW OF VIP - FIG 3



4.02.05

ELECTRIC MOUTH HIGH-LOW NUMBER GAME

0000	F800	B3B4	F803	B6F8	04B7	B2F8	01BB	F860
0010	ABF8	FFA2	F829	A3F8	E0A4	F881	BCB1	F895
0020	ACF8	46A1	F802	B5E2	D3F8	6354	3638	04FF
0030	0154	FB00	3229	302C	14E3	C464	FC66	4737
0040	3F66	2837	4366	4537	4766	5A37	4B66	4537
0050	4F66	2037	5366	4537	5766	7037	5B66	4537
0060	5F66	5937	6366	4537	6766	8937	6B66	4537
0070	6F66	1F37	7366	4537	7766	4537	7B66	3C37
0080	7F66	4537	8366	6437	8766	4537	8B66	8937
0090	8F66	4537	9366	0137	9766	4537	9B66	1C37
00A0	9F66	4737	A366	4737	A766	4737	AB66	5A37
00B0	AF66	4537	B366	2037	B766	4537	BB66	5637
00C0	BF66	4537	C366	3D37	C766	4537	CB66	6137
00D0	CFE2	69DC	FEFE	FEFE	54DC	04F4	30E3	0000
00E0	5019	00A5	0554	2430	F000	0000	0000	0000
00F0	64FB	61C0	0100	0000	0000	0000	0000	0000

0100	307A	1404	24E4	F532	1033	F630	F914	3090
0110	E364	FC66	4737	1566	2E37	1966	4537	1D66
0120	2A37	2166	4737	2566	4737	2966	3437	2D66
0130	4537	3166	3137	3566	4537	3966	8037	3D66
0140	4737	4166	4737	4566	8A37	4966	4537	4D66
0150	7037	5166	4537	5566	6037	5966	4537	5D66
0160	1437	6166	4537	6566	0537	69F8	00B0	A0E0
0170	D000	0000	0000	0000	0000	F860	AB04	A606
0180	A707	5B17	1B1B	1B1B	1B1B	1B1B	075B	3002
0190	F8BC	AB04	A606	A707	5B17	1B1B	1B1B	1B1B
01A0	1B1B	075B	E364	FC66	4737	A966	8A37	AD66
01B0	4537	B166	7037	B566	4537	B966	1337	BD66
01C0	4537	C166	4537	C566	4537	C966	6037	CD66
01D0	4537	D166	0237	D566	4537	D966	6737	DD66
01E0	4537	E166	4537	E566	8C37	E966	4537	ED66
01F0	3A37	F1C0	00D1	C003	9AC0	03A3	0000	0000

0200	0001	0203	0405	0607	0809	0000	0000	0000
0210	0A0B	0C0D	0E0F	1011	1213	0000	0000	0000
0220	1415	1617	1819	1A1B	1C1D	0000	0000	0000
0230	1E1F	2021	2223	2425	2627	0000	0000	0000
0240	2829	2A2B	2C2D	2E2F	3031	0000	0000	0000
0250	3233	3435	3637	3839	3A3B	0000	0000	0000
0260	3C3D	3E3F	4041	4243	4445	0000	0000	0000
0270	4647	4849	4A4B	4C4D	4E4F	0000	0000	0000
0280	5051	5253	5455	5657	5859	0000	0000	0000
0290	5A5B	5C5D	5E5F	6061	6263	0000	0000	0000

ELECTRIC MOUTH HIGH-LOW NUMBER GAME (Cont'd)

0300	0000	0204	0608	0A0C	0E10	1214	1618	1A1C
0310	1E20	2224	2628	2A2C	2E30	3234	3638	3A3C
0320	3E40	4244	4648	4A4C	4E50	5254	5658	5A5C
0330	5E60	6264	6668	6A6C	6E70	7274	7678	7A7C
0340	7E80	8284	8688	8A8C	8E90	9294	9698	9A9C
0350	9EA0	A2A4	A6A8	AAAC	AEB0	B2B4	B6B8	BABC
0360	BEC0	C2C4	0000	0000	0000	0000	0000	0000
0370	0000	0000	0000	0000	0000	0000	0000	0000
0380	0000	0000	0000	0000	0000	0000	0000	0000
0390	0000	0000	0000	0000	0000	F8DC	ABF8	675B
03A0	C001	0DF8	DC4B	F85B	5BC0	010D	0000	0000

0400	0145	0245	0345	0445	0545	0645	0745	0845
0410	0945	0A45	0B45	0C45	0D45	0E45	0F45	1045
0420	1145	1245	1345	1445	1401	1402	1403	1404
0430	1405	1406	1407	1408	1409	1545	1501	1502
0440	1503	1504	1505	1506	1507	1508	1509	1645
0450	1601	1602	1603	1604	1605	1606	1607	1608
0460	1609	1745	1701	1702	1703	1704	1705	1706
0470	1707	1708	1709	1845	1801	1802	1803	1804
0480	1805	1806	1807	1808	1809	1945	1901	1902
0490	1903	1904	1905	1906	1907	1908	1909	1A45
04A0	1A01	1A02	1A03	1A04	1A05	1A06	1A07	1A08
04B0	1A09	1B45	1B01	1B02	1B03	1B04	1B05	1B06
04C0	1B07	1B08	1B09	0000	0C00	0000	0000	0000

After loading the program (5 pages), set RUN/RES switch to RUN. Depress hex key C to start. Input number guesses via hex keyboard in the format 01 thru 99. When number is guessed correctly, depressing the last key used on the hex keyboard will restart the game without having to reset.

Tom Swan
P.O. Box 1014
Columbia, MD 21044

19-May-82

PAY AUTHORS? WHAT FOR...?

Dear everyone,

I am glad to finally see this issue of paying authors out in the open. Good. It is about time, and I applaud Ray for taking the bull by the horns. This is one bull fight I watch with great interest.

Here we go.

I firmly and emphatically cast my vote to pay authors for their work. Period. If this comes to ten cents per page, fine, but some fee should be paid to all contributors. Not a gift; a fee. There, I said it. I should have said it long ago.

Some readers will probably be thinking, 'oh, that's ok for him to say, he stands to gain after all.' I feel as the politician caught between his ideals and his business deals must think, 'How did I get into this mess?' Let me tell you.

The first year I decided to begin writing (again, but that's another story) I brought in $75. I worked eight hours a day for a full year, framed the only check I received, and was thrilled with my imminent success. The next year I did a little better. $110. We moved to Mexico where I felt I could continue to work. A good friend helped us afford the move and employed us for a while in his business in Taxco. Anne managed to feed the two of us.

Then I found a subject I felt comfortable with. I had been doing some reading. Small computers seemed new, fresh, exciting -- I felt and still feel these emotions -- and I started writing about computers as I learned what was going on. I couldn't help but gush with enthusiasm, and the Viper offered an easy way for a writer to rub on some much needed polish. It was the best proving ground I could have hoped for.

Things began to look brighter. The Viper was running two and sometimes three of my articles in each issue. I was writing something like 10 articles a week. The excess material was culled into the Pips book series which brought in a little money. Some of the left over material is still showing up in the new Viper to my constant surprise and gratification. I have a box full of things never seen by anyone -- I don't even know what's in that Pandoric crate and I'm afraid to open it. As I look back on that flow of material, I remember it was coming out of my ears, and some of it, when I read through the old issues, sounds as though it originated from other bodily orifices, but some of the articles still make me feel proud to have played a part.

And that's the point. Pride. Satisfaction. Self esteem. At the height of the flow of those early issues I requested postage money -- I asked for $15 per article (I would have accepted $5) -- and was turned down by ARESCO. I never told anyone, but at that time, after publisher Terry Lauderau said she would never pay authors, I felt crushed, devastated, used, and hurt. I sent ARESCO a letter which was never published. My resignation. I would never again contribute to the Viper. I was sorry and I would miss the work and I hoped I would be missed but I really could not continue to work for free. I had no other income -- nothing. We were at the end of the rope, out of money in a foreign place. Continuing made no sense. The work was consuming my time and I was about to be consumed by the work.

I gave up, drank more tequila for a week or so than I needed, came out of it somehow, took a deep breath and wrote the programming manual for Hayden in three hectic weeks. They bought it. Then ARESCO, out of desperation I now believe, hired me to edit their newsletters, the Viper and the Rainbow. We knew, suspected anyway, that the newsletters were dying, but all of a sudden it looked as though we would survive. We did, but the newsletters were soon cancelled and we had to leave our heaven in the hills and move back to the USA and look for jobs.

Why do I tell you all this? So you don't make the same mistakes all over again, not for my sake so much, but for the new writers holding their breaths that someone will publish and maybe even pay for what they work so hard to produce. So you realize that this issue over paying authors is really a decision point about the kind of publication you want the Viper to be.

Let's talk turkey. Magazines pay freelance authors for selfish reasons. It is cheaper than hiring full time writers to sit around twiddling their word processors. Also, editors realize that competition will produce better writing, better research, more diversity, and a better magazine all around. It works.

The same policy will work for us, and will insure the quality and, yes, the survival of the Viper. We stand only to gain by paying authors for their work. Please vote "yes" for this new policy.

With the best regards,

Tom Swan

Tom Swan

(Ed. P.S. 55% of you said "yes", 34% "didn't care" either way, 11% said No,)
- on the questionaire)

Ray

RUSSIAN ROULETTE
by
DAVID RUTH

I didn't actually write this game. I got it out of Volume 1, Issue 6 of the VIPER. The game was really written by Carmelo Cortez, I just added more graphics. You play the game the same way. Press any key to pull the trigger. Get ten "Clicks" in a row and you win. This game uses CHIP-8.

```
0200  650A 2288 22BA C307 4304 1232 A25A 6110
0210  D135 A25F 6118 D135 A264 6120 D135 6430
0220  F415 F407 3400 1222 00E0 75FF 4500 1246
0230  1202 A269 6110 D135 A26E 6118 D135 A273
0240  6120 D135 12C8 A278 6110 D135 A27D 6118
0250  D135 A282 6120 D135 1258 E888 8888 EEEE
0260  4848 48EE AACA CAC0 AAF7 5577 55F5 6565
0270  554D 4DD5 1515 40D5 8B89 89A9 DBB2 322A
0280  26A2 AAAA AA00 AA20 A29E 6626 6710 D675
0290  7608 A2A3 D679 7608 A2AC D67D 00EE 0020
02A0  20FF FF00 000F FFFF 0101 0101 0000 FFFF
02B0  FF5F 5F1F FF1F 1F1F 1F00 F20A 6E37 6F15
02C0  A2C6 DEF2 00EE 8080 22DE 6A25 6B13 A2C6
02D0  DAB1 DAB1 7AFF 3A04 12D0 DAB1 12F0 A2E8
02E0  6C00 6D12 DCD8 00EE 3C7E 81A5 8199 423C
02F0  6A64 FA18 A200 6B00 6D12 DBD8 6E60 FE15

0300  FE07 3E00 1300 00E0 6A10 6B0E A330 DAB4
0310  7A08 A334 DAB4 7A08 A338 DAB4 7A08 A33C
0320  DAB4 6C40 FC15 FC07 3C00 1326 00E0 1200
0330  EE4A 4C4A AE4A 4E4A EE8A AEEA E94D 4BE9
0340  0000 0000
```

BOMBS AWAY IN COLOR AND SOUND

Step 1: Install the VIP Color Board and/or the VIP Simple Sound Board.

Step 2: Load the CHIP-8X interpreter.

Step 3: Load the following:

```
0300   02A0 02A0 2434 242A A3D0 6B00 6C1A DBC2
0310   A3D4 643C 6606 D463 6700 6819 23A2 23AC
0320   4800 13D8 6509 A3D7 6300 6D05 EDA1 6301
0330   8E40 EDA1 DE51 133C EDA1 23D8 A3D4 D463
0340   1342 74FF D463 A3D0 DBC2 CD04 8BD1 DBC2
0350   3F00 1392 A3CD D9A2 CD08 4D00 7903 79FD
0360   D9A2 3F00 138C 4300 132A A3D7 DE51 451F
0370   1386 7502 2480 DE51 3F01 133C 6D1f 8D52
0380   4D1F 138C 1392 23AC 78FF 131E 23A2 7705
0390   1396 23A2 770A 23A2 6180 248A A3D7 DE51
03A0   1386 A3F8 F733 6300 23B6 00EE A3F8 F833
03B0   6332 23B6 00EE 6D00 F265 F029 D3D5 7305
03C0   F129 D3D5 7305 F229 D3D5 00EE 0108 7F7C
03D0   083E 6008 183C FF08 A400 6311 6D0B D3D5
03E0   A405 6319 D3D5 A40A 6323 D3D5 A40F 632B
03F0   D3D5 6300 1420 0000 0000 0000 0000 0000


0400   EE8A 8AAA EEEF A5A5 A5EF 7A2A 3B29 79BA
0410   A2B2 203A 343A 3CD6 541C 0C40 9E25 680C
0420   6E0A EE9E 1420 00E0 1306 A3CD 6938 6A1E
0430   D9A2 00EE 6000 6100 6200 B109 7104 3164
0440   143A 6002 6100 6209 B10F 7104 3164 1448
0450   6100 6218 B102 7104 3164 1454 6003 6100
0460   621A B104 7104 3164 1462 6006 6100 621E
0470   B102 7104 3164 1470 00EE 00EE 6100 6218
0480   6003 F1F8 F018 71F3 00EE 6010 F1F8 F018
0490   00EE 0000 0000 0000
```

To Play: Key 5 drops the bomb. The small sub is worth 10 points and the large sub is worth 5 points. Key A resets game faster than toggle switch.

ERROR TRAP

The Mini-Calculator program in VIPER 4.01 has some code missing. You'll notice that memory location 0232 contains 0630, a machine language subroutine, and there is no code at 0630. Memory location 0288 contains 0650, another call to a non-existing machine code routine. Evidently, the missing sections of code were "sliced off" the end of the program, since they would have been the last items in the program. Very sorry about that one, folks, but we'll try to find the missing code and print it in VIPER as soon as possible.

STAR-SPANGLED BANNER

Step 1: Load the PIN-8 interpreter.

Step 2: Load the following:

```
0259    BF
02E0    0202 0202 0303 0303 0404 0404 0505 0505
0300    0105 070B 0D11 1317 0105 070B 0D11 1317
0310    1A1E 2024 2629 2B2F 3134 383C 3F40 4447
0320    4B00 0000 0000 0000 0000 0000 0000 0000
0330-037F    0000
0380    0105 070B 0D11 1317 0105 070B 0D11 1317
0390    1A1E 2024 2629 2B2F 3134 373B 3D3E 4144
03A0    4800 0000 0000 0000 0000 0000 0000 0000
03B0-03FF    0000
0400    0049 0662 6669 AE52 106E 6668 A929 2992
0410    306E AD2B 2D6E 6E69 6662 3232 7273 75B5
0420    3332 7072 73B3 7392 306E AD2B 2D6E 6668
0430    A969 6E6E 2E2D 6B6B 6B70 3332 302E AECD
0440    2929 8E30 3233 B52E 3092 3370 AE00 0000
0450-04FF    0000
0500    0049 0662 6669 A646 0666 6664 A429 2989
0510    2969 A828 2868 6E69 6662 2E2E 6E70 72B2
0520    302E 6D6E 70B0 6989 2969 A929 2969 6664
0530    A469 6667 6967 6769 6727 296B A9C0 2929
0540    A929 2EAE 2B28 892A 67A6 0000 0000 0000
0550-05FF    0000
0600-06FE    0000
06FF    ED
```

Break Table:

```
0270    1280 E016 80E0 FE12 80E0 1680 E0FE 1280
0280    E016 80E0 FF00 0000
```

Step 3: Store on tape 7 pages.

MACHINE CODE Part 6

P. V. Piescik, 157 Charter Rd., Wethersfield, CT 06109

Last time I stuck in the code for DRVROUT-parallel, and here's the discussion:

First we check the handshake signal from the device to avoid sending data to a device which is busy, off-line, powered-down, etc. Some devices don't provide handshaking (ugh!); we'll talk about them later. You'll have to check the manual for your device to find what the signal is, its meaning, whether it's normally high or low, and how it fits into the data cycle of the device. We'll assume here that the signal is called BUSY or NRD (Not Ready for Data), uses negative logic (normally high, goes low for busy). We will assume that data set-up times, etc., are handled by the data port/latch hardware. This signal does not acknowledge that data has been accepted by the device. We'll assume that EF4 is used for this signal; if EF4 is low (true), we wait. Since the VIP has port 3 on-board, we'll use it. If you are using other EF's or ports, you'll have a couple of bytes to change.

```
0036          DRVROUT:  ORG *
0036  37 36             B4 *          ..wait here while busy
0038  22 52             DEC 2;STR 2   ..data from D to stack
003A  63                OUT 3         ..output to port 3
003B  D5                SEP 5         ..return
```

The OUT instruction increments R(X), so the stack pointer is left where we found it. We decrement it first so the increment won't point it into live data even momentarily.

If the device is serial but the serialization of the data is done by a UART, the device which is of interest to the software is the UART! It's interfaced in parallel through two ports: one for data and another for status. Usually, the MSB of the status byte is TBMT (Transmit Buffer eMpTy); if this bit is 0, we wait. Let's assume that port 6-in is status and port 6-out is data.

```
0036          DRVROUT:  ORG *
0036  6E FE             INP 6;SHL     ..read status, TBMT to DF
0038  3B 36             BNF DRVROUT   ..wait until TBMT=1
003A  22 52             DEC 2;STR 2   ..data from D to stack
003C  66                OUT 6         ..output to port 6
003D  D5                SEP 5
```

The headache come when a nasty device (TTY, Netronics VID, etc.) doesn't provide any handshaking. We have to "time out" the data cycle for each byte we send to the device. If we're using serial software, we're busy the whole time anyway. However, special functions which take more than one byte-cycle to complete are a problem. Carriage-return on a TTY and a lot of printers takes about 0.5 sec. (5 chars at 110 baud); the VID and other video boards have functions to clear the screen, part of the screen, and /or part of a line. These operations can take 400 ms. or more. With such a device, you have to intercept each code you send, determine if an additional delay is needed, and then kill time. You can either sit in a delay loop like BAUD, or send enough NUL's to fill the time. If you send the next character too soon, the device will either miss it entirely or interpret it as garbage.

We saved memory by storing only CR at the end of a line, instead of CR, LF and some number of NULs. We must output the LF and NULs by calling PUTCR. PUTCR will send enough NULs to take care of a TTY. It's also easily modified to send more or fewer NULs.

```
0063                PUTCR: ORG *
0063  F8 0D                LDI #0D             ..generate ASCII CR
0065  D4 00 40 V           SEP 4,A(DRVROUT)    ..call via SCRT
0068  F8 0A                LDI #0A             ..generate ASCII LF
006A  D4 00 40 V           SEP 4,A(DRVROUT)
006D  F8 05                LDI 5               ..how many NULs
006F                PTCR1: ORG *
006F  73                   STXD                ..push counter
0070  F8 00                LDI 0               ..ASCII NUL
0072  D4 00 40 V           SEP 4,A(DRVROUT)
0075  60 F0                IRX;LDX             ..pop counter
0077  FF 01                SMI 1               ..decrement counter
0079  3A 6F    R           BNZ PTCR1           ..loop until 0
007B  D5                   SEP 5
```

The "V"s and "R" flying in the listing indicate "external" and "relocatable" references, respectively. If your device is parallel, you don't need BAUD, and could put DRVROUT at 0036-003B. In that case, A(DRVROUT)=0036, not 0040 as above; the "V" tells you that a value is dependent on a location which is not part of this routine. As you'd continue, PUTCR would go at 003C-0054 instead of 0063-007B as listed above. PTCR1 would not be at 006F, and the branch instruction flagged with the "R" would have to be changed to reflect the relocation of this routine. If you were using an Assembler Program, it would take care of these references for you, provided that the defintions of the labels were assembled at the same time as the references. Just remember that "V" flags a value which does not change if this routine moves; it's dependent on another routine's location. "R" flags a value that moves with this routine, and is independent of the location of any other routine.

To relocate "R" values, calculate a relocation factor for the entire routine by subtracting the listing address from the actual address. If we use this listing, but PUTCR is at 003C, the relo factor = 003C - 0063 = FFD9. This relo factor is <u>added</u> to the address value flagged by "R": 006F + FFD9 = 0048. Always calculate the full address, then use the low-order byte for short branches and the entire address for long branches. We'll also see relocatable values appearing as immediate data, and may have to use the high-order byte. I'll list routines to allow for use of the longest version of each of the previous modules.

PUTLINE will complete the output half of this project. We'll be matching data structures (step 2), formats (step 3), using the algorithm (step 4, on pg. 4.01.15).

```
007C                  PUTLINE: ORG *
007C  32 9F      R             BZ PTLNX            ..return if size=0
007E  87 73                    GLO 7;STXD          ..save buffer address
0080  97 73                    GHI 7;STXD          ..   R7
0082  9F FF 01                 GHI F;SMI 1         ..loop for size-1
0085                  PTLN1:   ORG *               ..  chars unless CR
0085  73                       STXD                ..push counter
0086  07                       LDN 7               ..get next char
0087  FB 0D                    XRI #0D             ..  is it CR?
0089  32 96      R             BZ PTLNCR           ..  yes, break loop
008B  47                       LDA 7               ..get char, bump ptr
008C  D4 00 40   V             SEP 4,A(DRVROUT)    ..send char
008F  60 F0                    IRX;LDX             ..pop counter
0091  FF 01                    SMI 1               ..decrement counter
0093  3A 85      R             BNZ PTLN1           ..loop until 0
0095  38                       SKP                 ..skip IRX unless
0096                  PTLNCR:  ORG *               ..  CR found early
0096  60                       IRX                 ..pop counter
0097  D4 00 63   V             SEP 4,A(PUTCR)      ..send CR LF NULs
009A  60                       IRX
009B  72 B7                    LDXA;PHI 7          ..restore R7
009D  F0 A7                    LDX;PLO 7           ..  buffer address
009F                  PTLNX:   ORG *
009F  D5                       SEP 5
```

If the user calls us with a buffer size = 0, we return without doing anything! After saving R7, we loop for size-1 characters, since we'll force CR as the size-th character anyway. Within the loop we check for CR without bumping the pointer, so we don't have to back up if it's not CR. The XRI destroys the char and we have to pick it up again. If it is CR, we don't need the pointer anymore and will be replacing it with the buffer address shortly. We have two exits from the loop (on CR or on count), and the loop counter is on the stack if we exit on CR. If we exit on count the counter is no longer on the stack. PTLNCR pops the stack in case we got there early, but we don't want to pop if we drop through after the loop--SKP at 0095.

For a test of PUTLINE, PUTCR and DRVROUT, use the following temporary routine:

```
00A0  F8 00 B7     LDI 0;PHI 7        ..this page
00A3  F8 AC     R  LDI A.0(TEXT)      ..buffer address
00A5  A7           PLO 7              ..  in R7
00A6  F8 13        LDI #13            ..buffer size in D
00A8  D4 00 7C V   SEP 4,A(PUTLINE)
00AB  23           DEC 3              ..halt
```

```
00AC                    TEXT: ORG *   ..mystery message
00AC  54 48 45 20
00B0  4D 4F 4E 53  54 45 52 20  4C 49 56 45  53 21 0D xx
00C0
```

This is a main routine. Insert the address at 0010 (A.1) and 0013 (A.0). It stops dead at 00AB, since it has no caller and has nowhere to return.

During the first draft of this issue, I found that we have enough code kicking around to get confusing, especially since it's spread throughout several issues. Let's nip this growing problem in the bud. What we need is a Storage Map to tell us which routines are in memory, where they are loaded, where the entry point is, and maybe, the size. It can be formatted like this:

```
LOAD ENTR SIZE  NAME______
0000 0000 0015  INIT
0015 0016 0012  CALL
0027 0028 000F  RET
0036 0037 000A  BAUD
0040 0040 0023  DRVROUT
0063 0063 0019  PUTCR
007C 007C 0024  PUTLINE
00A0            (next avail. loc. w/o temp. test)
```

GETLINE. Review the earlier discussions of the overall problem, user convenience, and input on pp. 4.01.13 - 15. I have one more thought about user convenience, for DOCAN. We're planning to echo CR to start a fresh line if the user CANs the one he's on, which is OK at the time, but it's confusing to re-read the terminal output. It looks like part of the same line was entered twice. Let us indicate that a line was CAN-d by first sending a backslash (ASCII 5C) then CR. This does not affect the reset of the buffer pointer and char count.

I'll answer the questions I posed on p. 4.01.15. If the input has DEL right after CAN, or more DELs than chars in the buffer, what do we do, and why? At some point, the DEL will arrive when the buffer is empty, so we'll ignore it! If we performed the usual DEL function, the char count would get very large (unsigned; if signed it would go negative) and won't be correct. More seriously, the buffer pointer would be decremented past the start of the buffer and storing the next char would clobber something.

If we have n-1 chars without a CR, and we'll force CR as the nth char anyway, can we force the CR before the next key is entered? NO!! We don't do anything early, at least, not without the user. He may have made an error and the next key could be CAN or DEL. If we force the CR early, we neither allow his correction, nor do we send correct input to his program when what he wanted was to end up with fewer chars in the buffer! The char count is the number of chars stored in the buffer, not keystrokes. With a few CANs and DELs, we could look at a lot more than n keystrokes.

GETLINE. 1) Specify the problem. Given a buffer address and size (n), input up to n chars from the keyboard via DRVRIN. Detect special chars: CR, CAN, DEL. For CR, store CR in the buffer and include it in the count; echo via PUTCR and terminate input. For CAN, reset the char count and buffer pointer, echo backslash CR, and continue. For DEL, decrement count and buffer pointer unless count is already 0, then continue. For all other chars, store at buffer pointer location, increment count and pointer, and continue. If n chars are input without detecting CR, replace the nth non-special char with CR, echo CR via PUTCR, and terminate input. Upon termination, restore buffer address and replace the size with the char count.

2) Data Structures. ASCII codes w/o parity; special codes: CR, CAN, DEL; buffer address; buffer size; char count.

3) Formats. ASCII w/o parity is 8 bits w/MSB=0, received in D and stored in buffer (except special); special codes are immediate data constants: CR=0D, CAN=18, DEL=7F for checking input; buffer address is 16 bits unsigned in R7; size is 8 bits unsigned in D (upon entry); count is 8 bits unsigned in D (upon return). NOTE: we'll decide where to keep the working char count and original buffer address later, since they probably will be needed by the subroutines.

4) Algorithm. Check buffer size, if 0, return. Else save buffer address (R7). For n chars (n=size), read keyboard via DRVRIN and check for special characters. Store non-special chars in buffer and increment count and pointer. For special chars, call DOCAN or DODEL; or for CR, store CR in buffer and increment count, echo via PUTCR and terminate. Force CR as nth char unless nth char is special. Upon termination, restore buffer address and return count in place of size, then return.

5) Modularity. Special char functions by DOCAN, DODEL; input by DRVRIN; echo by PUTCR and DRVROUT.

DOCAN. 1) Specify the problem. Echo backslash then CR (via PUT-CR), reset buffer pointer to start of buffer, reset char count to 0.

2) Data Structures. Constant backslash = 5C; buffer pointer; char count.

3) Formats. Backslash is immediate data; buffer pointer is R7; char count is RE.0.

4) Algorithm. Generate backslash and echo via DRVROUT; echo CR via PUTCR. Subtract count from pointer to reset pointer, set count to 0. Return.

5) Modularity. None.

DODEL. 1) Specify problem. Delete last char from buffer, erase from screen (or indication deletion on hardcopy with backslash), decrement buffer pointer and char count. Ignore DEL if buffer is already empty (count = 0). This is device-dependent with respect to echo: some video boards have DEL function; others require BS SP BS; TTY gets backslash.

2) Data Structures. Buffer pointer; char count; constants: BS=08, SP=20, backslash=5C, DEL=7F (as needed).

3) Formats. Buffer pointer is R7; char count is RE.0; constants are immediate data.

4) Algorithm. Check char count, if 0, return. Else decrement pointer and count. Generate and send device-dependent echo.

5) Modularity. None.

NOTE: The contents of memory in the buffer are not changed. The buffer memory is accessible only through the buffer pointer and char count, so the deleted char is now out of reach.

DRVRIN. Like DRVROUT, this is device-dependent and we'll cover 3 situations: a) parallel device, b) serial device with UART, and c) serial device with software serialization.

DRVRIN-parallel. 1) Problem. Input a char, when available, from the keyboard through a port and return it in D (SCRT makes a copy in RF.1), stripped of parity (MSB=0).

2) Data Structures. ASCII code w/parity; ASCII code w/o parity.

3) Formats. ASCII w/parity is 8 bits in D; ASCII w/o parity is 8 bits in D w/MSB=0.

4) Algorithm. Check handshake until data is available. Input ASCII w/parity through the port. Mask off MSB. Check handshake until key is released, if necessary.

5) Modularity. None.

Whether we'll wait for the key to be released depends on how the keypress/data available signal is presented. Some keyboards give a one-shot pulse when a key is pressed; others give a steady signal for as long as the key is held down. If we check for key release with the pulsed signal, no harm is done. However, if we don't check with the steady signal, the program will get back to DRVRIN several times before the user can lift his finger(!) and we will get multiple reads on each character.

DRVRIN-serial. 1) Problem. Input a char, bit-by-bit, LSB first, from the terminal and return it in D (SCRT makes a copy in RF.1), stripped of parity (MSB=0). All bits must be read at the baud rate of the device, and the start-bit should be checked for validity (i.e., reject noise).

2) Data Structures. Start-, data-, parity-, and stop-bits; ASCII w/parity; ASCII w/o parity; byte-in-progress (b-i-p); bit-count; half-time constant.

3) Formats. Bits are 1 bit, interpreted from the input signal; data-bits may be created in DF and shifted, or in immediate data and OR'd, into the b-i-p; b-i-p is 8 bits in RF.1; bit-count is 8 bits unsigned in RF.0.

4) Algorithm. Check input until line changes from idle condition; time out for ½ bit-time at baud rate, the recheck line. If idle again, reject noise and continue checking. Else read 8 bits, LSB first, to form b-i-p. Mask off parity bit w/o checking and return in D. Reading stop-bits is optional.

5) Modularity. None.

We want to read each bit at the center of its time-slice to stay away from transitions in the line between bits. Since we'll catch the leading edge of the start-bit, this is why we delay for ½ bit-time. We DO get the leading edge--we wait in a 1-instruction loop (9 usec.) and we can handle 110-4800 baud with the DRVRIN/OUT and BAUD combination. A bit-time is 208 usec. at 4800 and 9091 usec. at 110. Noise tends to last 1000 usec. or less, which is shorter than ½ bit-time if we're under 500 baud. Your keyboard is probably 110 or 300 baud, so we'll check. At higher baud rates, our checking may be ineffective; the hardware must prevent noise.

Reading bits is a little strange! We don't actually have a bit anywhere in the 1802 which can be used as data, as a result of the EF-line. We capture the line condition at one instant during the execution of a short branch (34-37 or 3C-3F). The code at the destination of the branch must generate a 1-bit or a 0-bit. We can create a bit in DF and shift it into the b-i-p; we'll have a '1' in DF after BAUD and can RSHR it into the b-i-p or SHR in a '0'; we can SHR the b-i-p first (shifts in '0') and OR in a '1' if we need it.

We don't have to read stop-bits! They're there to allow us time to complete our cycle, and we know when we've read 8 bits. Since our cycle also includes other activities, like having GETLINE store the byte and echo it, we may be able to get some/all of that done during the stop-bits.

I don't use an automatic echo. It screws up special functions like CR, CAN and DEL; unless you can type over 50 wpm on a device that is only 110 baud, I can keep up even with a separate echo.

Let's start coding with DRVRIN-serial; BAUD already exists.

```
00A0                DRVRIN:  ORG *
00A0  22 61                  DEC 2;OUT 1  ..TV off
00A2  87 73                  GLO 7;STXD   ..save R7 for use
00A4  97 73                  GHI 7;STXD   ..  as BAUD PC
00A6  F8 08 AF               LDI 8;PLO F  ..bit counter
00A9  93 B7                  GHI 3;PHI 7  ..BAUD on same page
00AB                DRVRIN1: ORG *           ..read start-bit
00AB  F8 39      V           LDI A.0(BAUD+2) ..kluge entry to over-
00AD  A7                     PLO 7           ..ride BAUD with a
00AE  F8 --                  LDI #--      ..½ bit-time constant
00B0  3F B0      R           BN4 *        ..wait while idle
00B2  D7                     SEP 7        ..wait ½ bit-time in BAUD
00B3  3F AB      R           BN4 DRVRIN1  ..reject noise
00B5                DRVRIN2: ORG *        ..read data-bits
00B5  D7                     SEP 7        ..wait 1 bit-time in BAUD
00B6  9F F6                  GHI F;SHR    ..get old b-i-p and make
00B8                ..room for new bit in MSB, MSB=0 after SHR
00B8  37 BC      R           B4 *+4       ..go for 0-bit
00BA  F9 80                  ORI #80      ..else jam 1-bit
00BC  BF                     PHI F        ..replace new b-i-p
00BD  C4                     NOP          ..stretch time to match
00BE                ..19-21 cy. loop in DRVROUT
00BE  2F 8F                  DEC F;GLO F  ..decrement bit counter
00C0  3A B5      R           BNZ DRVRIN2  ..and loop until 0
00C2  60                     IRX          ..prime pop
00C3  72 B7                  LDXA;PHI 7   ..restore R7
00C5  F0 A7                  LDX;PLO 7
00C7  9F FA 7F               GHI F;ANI #7F ..strip parity
00CA  69                     INP 1         ..TV on (optional)
00CB  D5                     SEP 5
```

In the DRVRIN1 loop, we have to set up R7.0 after rejecting noise; BAUD will have left R7 at the normal entry address. The time constant at 00AF is simply ½ the constant in BAUD. We'll be about 2 cy. long for ½ bit-time, but it's OK--only 9% off even at 9600.

In the DRVRIN2 loop, we get the old b-i-p and shift it right to put a '0' in the MSB before we know what the new bit is; if we get a '0' we're done, else we OR in a '1'. CAUTION: The branches at 00B0, 00B3, 00B8 may need 37's for 3F's or vice-versa, depending on whether your interface is true RS232 (neg. logic) or is pos. logic to make life easy for the pull-up resistor on the EF-line. I've done this one backwards a couple of hundred times!!

Just before we leave, we mask off the parity-bit by AND-ing it with a mask which has a '0' MSB and all other bits are '1' to save them as they are.

Intoxication Tester

by Nicholas N. Panasis

The length of time that the random number is shown on the screen is controlled by changing location 026E. The number of digits missed determines the message you receive on the screen. You are allowed to miss one digit and still be considered "sober!"

0200	6001	6107	6206	6300	A3FA	F21E	D015	7008
0210	7301	3308	120A	6010	6111	6300	F21E	D015
0220	7008	7301	3304	121C	6080	F018	61D0	F115
0230	F107	3100	1230	00E0	C00F	C10F	C20F	C30F
0240	C40F	C50F	A550	F555	F029	660B	6709	D675
0250	7607	F129	D675	7607	F229	D675	7607	F329
0260	D675	7607	F429	D675	7607	F529	D675	6880
0270	F815	F807	3800	1272	00E0	6700	F60A	5060
0280	7702	7700	F60A	5160	7702	7700	F60A	5260
0290	7702	7700	F60A	5360	7702	7700	F60A	5460
02A0	7702	7700	F60A	5560	7702	7700	8070	B2B0
02B0	12BE	12BE	12DA	12DA	12F6	12F6	1312	6113
02C0	620D	6306	6400	A442	F31E	D125	7108	7401
02D0	3403	12C8	6F80	FF18	12D8	6107	620D	6306
02E0	6400	A454	F31E	D125	7108	7401	3407	12E4
02F0	6F80	FF18	12F4	6113	620D	6306	6400	A47E
0300	F31E	D125	7108	7401	3404	1300	6F80	FF18
0310	1310	610F	620D	6306	6400	A496	F31E	D125
0320	7108	7401	3405	131C	6F80	FF18	132C	0000
0400	E84C	4A49	E800	BE88	8888	8800	F492	9192
0410	F400	5D89	0989	5D00	EF09	0F09	E900	7D10
0420	1010	1100	DE92	9292	DE00	88C8	A898	8800
0430	FB22	2322	2300	CE10	8C02	DC00	FB22	2322
0440	2300	DC12	9C14	D200	7784	6414	E700	BD95
0450	9D95	BD00	EE09	CE0A	E900	9794	F794	9400
0460	A1A1	A1A1	BD00	E000	C000	0000	7B2A	2B2A
0470	7A00	9252	9292	5E00	8ACA	AB9A	8A00	4088
0480	0080	4000	F754	5755	F400	25A5	2525	BD00
0490	1495	5635	1400	8000	0000	8000	7686	6514
04A0	E400	DED2	5E52	5200	7484	6714	E400	BDA0
04B0	B8A0	BD00	E0A0	A0A0	E000	0000	0000	0000

VIP Hobby Computer Assn.
32 Ainsworth Avenue
East Brunswick, N.J. 08816 USA

KILMER FACILITY, NJ 088
PM
JUL 27
1982

KILMER FACILITY, NJ 088
PM
JUL 27
1982

FIRST CLASS

A Final word:

Although RCA is no longer producing the VIP computer, there is still support for the 1802 by several other companies. For example, Quest has come out with a new unit, called the "Venture." You can get more info from the company, but a first look at their ad in various magazines, shows some very interesting specs. For example, the Venture's video display will permit up to 4096 user-defined characters or alphanumeric symbols, and graphics symbols. The Venture may be expanded with "full BASIC," 3 ROM monitors, assembler, etc. It will run video games, CHIP-8 programs, and all Quest 1802 software. Very interesting!

ARTICLES PLANNED for future issues:

VIP PIN-8 music by David Ruth

Upgrade your Color Board by Jeff Jones

Little Loops by Tom Swan