VIPER

VOLUME 2, ISSUE 2 AN ARESCO PUBLICATION AUGUST 1979, $2.00

TABLE OF CONTENTS

E D I T O R I A L

by Rick Simpson

At the CES show in Chicago in June, RCA showed their new VIP II for the first time. It's an interesting combination of the old and the new; for the old, it uses the 1802 processor and the 1862 color video chip used in the VIP Color board. It includes the Simple Sound circuitry and it has a ROM monitor very like the one in the 'old' VIP. The tape interface is the same, so all VIP CHIP-8 programs load and run without modification. But the new features are really impressive. The VIP II sports a full typewriter-style ASCII keyboard - as well as the familiar hexpad - and 14K of additional ROM containing a full BASIC interpreter. With additional color, sound, and plotting commands. A Built-in FCC-approved RF modulator. RAM increased to 8K. And the price for all this? Around $400. Indications are that it will be deliverable early next year.

The VIPER is published ten times per year by ARESCO Inc., at 6303 Golden Hook, Columbia MD 21044. Mailed to subscribers on the 15th day of each month except June and December. Single copy price is $2 and subscription price is $15 for all 10 issues of the current volume. Subscriptions do not carry over from one volume to the next. Subscribers who wish to order less than the full volume should send $2 for each issue desired. Renewals are accepted during the last two months of the current volume year, and the first issue of each volume is published in July.

Application to mail at second class postage rates is pending in Columbia MD 21045. POSTMASTER: Send all address changes to THE VIPER, Box 1142, Columbia MD 21044.

THE VIPER is an ARESCO Inc. publication. Editor: Terry Laudereau, Co-Editor: Rick Simpson. Corresponding editor is Tom Swan.

Readers are encouraged to submit articles of general interest to VIP owners for publication in the VIPER. Material submitted will be considered free of copyright restrictions. Those submissions received by the 1st day of a month will be considered for inclusion in that month's issue.

SUBSCRIPTION RATES:

USA residents: $15/10 issues. Non USA residents should include an additional $10 for Air Mail postage. If such postage is not included with the order, the newsletter will be sent second class with the remainder of the mailing. Purchase orders will not be accepted. If subscriber specifies UPS COD the first issue will be sent COD with the collectible amount to be $15 plus a $1 COD charge plus shipping costs. Personal checks, Master Charge, and VISA orders are accepted. Checks drawn on foreign banks should be payable in U.S. Dollars.

ADVERTISING RATES:

Classified Advertising: $10/3 lines of 60 characters each.

Page Rates:

Full Page	$85
Half Page	$45
Quarter Page	$25

Payment must accompany ads. Ads should be camera ready positive artwork. If our printer charges us extra for preparing your ad copy, we will bill you at cost + 10%.

Dealers are invited to request the ARESCO Inc. dealer information package by writing in care of Dealer Services.

Viper: I would like to see an article in the VIPER on HIRES (128 x 64) machine language video control. I can produce HIRES pictures, but only without motion, as I apparently don't fully understand the interrupt process. Also, I understand the TINY BASIC for the VIP will use the starting address 0000, and will not allow use of CHIP-8 (any mod), without unplugging and re-plugging components. How about a program to correct this? And a brief on machine language color control? Any enterprising soul working on an 8080 simulator? And how about recommending a "standard" joystick mod? Quick comment on the expansion keypad: not so good. It requires more effort than the original keypad; some keys require more pressure than others on the same keypad, and the exposed circuitry on the reverse needs to be covered. Still, overall acceptable. Maybe mine has lemon flavoring. But I will buy another - it allows more freedom of movement to both players. - Louis Schlekau III

Louis - While I'm not ready to swear to it, it seems to me that TINY BASIC comes up after RESET unless you press the E key - whereupon you come up in the monitor and can load CHIP-8 programs as usual. Machine language color control will no doubt happen when more people have the color board, and so far as I know, no one is working on an 8080 simulator. I'll check my files more carefully, however, and let you know for sure if someone is doing so. We're looking for a standard joystick mod, too. As people send in their ideas and suggestions, we'll certainly let everyone look them over (maybe have an issue devoted to the various suggestions), and see what we can come up with. - Terry

Rick - Since text editor programs are becoming available on the VIP, wouldn't it be possible to have a program to convert amateur radio code signals from a receiver to a word display on the VIP monitor? And how about using a keyboard to code the signals? These programs would have lots of appreciators! - C Thompson

C.T. - Sure, it's possible! Perhaps someone out there is doing just that at this very moment, and just needs this reminder to get his notes in order to write it up for us! - Rick

Rick - Just got my first nine issues of VIPER, and I'm really pleased. The thing that prompted me to subscribe was the mods to ELF II so it can run CHIP-8 and VIP games. Now we ELF owners can play with those clever CHIP-8 games. I made a few modifications you may be interested in. First, I didn't like the idea of using a calculator keyboard for the new keypad. Second, why scratch-build something that's already built - and cheap? I'm referring to the VP-580 keypad. All I did was remove the original ribbon cable with 7 connectors, and replace it with an 8 connector wire. Carefully cut pin one of the CD4515 pin from the board, cutting as close to the foil as possible. Lift the pin gently away from the board until the pin is horizontal, and connect the eighth wire to this pin. Be careful not to glop solder all over the place! This wire becomes your enable pin. Since I have a "Giant Board"tm I connected the enable pin to the 62 instruction. Connect up the remaining seven wires as per schematic in the VIP Instruction Manual. I also connected the output wire to EF3 instead of EF4.

By doing this, you eliminate most software changes in the operating system code. Change only locations 8001 and 8056. This goes for CHIP-8 also. Change only locations 000A, 010B, and 012A. I also moved the CHIP-8 patch from 0CF0 to 01F2. There is just enough room to do this. Jump to 01F2 to start CHIP-8 programs. It's also possible to move the CHIP-8 display page from 07 to 0B. Be warned, however, that by modifying the VP-580 keypad, you probably void its warranty. How would I obtain a single page copy (if it exists) of the schematic for the double graphics (128 x 64) shown in issue seven? Are the outputs of the two RAMS wireORed together? One last thing: Are there any VIP or ELF owners near me? If so, please have them write or call (215-LU3-8171). I'd like to get together and maybe form a user's group or something. - Al McCann, 30 S Elmwood Avenue, Glenolden, PA 19036

Al - Thanks for the information on the ELF mods. As for the schematic, we can't get a better one than is in #7. Rick says the RAMs are probably wireORed, but he suspects they're tri-state outputs. I can't translate that - I don't know hardware. Hope my second-hand answer helps! If not, perhaps by printing your name and address, someone who has the answer first-hand will call or write to you. - Terry

Viper - Here is yet another modification to the Sam Hersch Editor Program published in VIPER #4. This mod allows the use of the Udo Pernisz relocating routine and the Norman Elliot five-address scan routine within the same program. After you enter the Hersch program, with the Pernisz modification, make these changes:

```
at 0282 enter 340C
  *0284       1288
   0286       1348
   0288       3405
   028A       1206
   028C       13B0

at 03B0 enter 02D2
   03B2       02D2
   03B4       02D2
   03B6       02D2
   03B8       02D2
   03BA       1206
```

* If you enter this address last, you can use the Hersch program to modify itself! - Raymod C Sills

(Editor's note: Recently we received a large supply of comments from John B Sewell, who admitted that he had been collecting them rather than mailing them. There isn't any way to reproduce the letters in order (since none of them were dated), so we're just printing them as we read them.)

Rick - Two suggestions to fellow novices: 1 - always record a program on tape immediately after entering it by keyboard. Then try to run it. If (when?) it craters, it is much easier to find an error in the entered program than in the smoking ruins. 2 - if you are going to use a television/RF modulator instead of a

video monitor, give up and don't try to save money by building one. Spend $30 for a Sup-R-Mod. I tried 3 kits and 2 scratch-builts before I gave up. I hate 1802 Machine Language! Anyone who uses a machine language subroutine in an otherwise civilized CHIP-8 program should be required to justify in writing the reason for doing so, clearly label it, and explain in detail what it does. I would like to see in the VIPER articles, or perhaps a continuing column, on writing programs in CHIP-8 for the beginner. For example, how do you bounce a spot off a wall? How do you tell if a spot adjacent to the displayed spot is 1 or 0? With all the problems of transcribing and entering printed programs by keyboard, and I find that I can usually enter two pages of instructions in one try with no errors, I much prefer this method to purchasing pre-recorded cassettes: First, I record my programs on a reel-to-reel recorder, because none of the 4 recorders and 8 brands of tape cassette I've tried would load successfully. Second: I'm cheap. Having already paid for the VIPER, I'd like as many programs in it as possible, and as few extra purchases as possible. This must, of course, be consistent with good program writers making a buck, but I sincerely hope no one is trying to make a living writing programs for folks who buy $250 microcomputers. Regarding Vol 1 #10, the Lunar Lander corrections by Udo Pernisz, page 18. What does the data at 0490 do? The program seems to work the same with or without it. Also, I presume the Lunar Lander that Neil Weigand said (on page 3) was poorly documented must not be the one by Udo, since his program is the best documented and annotated that I have ever seen. It is much, much, MUCH better than those in the Instruction Manual, VIP-311. Only because of the good documentation was I able to find and increase the amount of fuel so that I could land successfully. And regarding the letter from D L Hartley, on page 6: Sympathy. I've been trying for only six months, but I know the feeling. The suggestion by Joe (page 16) about good guys to help novices is very good.(For "novice", read "dummy".) Best for me would be somebody in Houston that I could bug by phone. Often the bug in a program can be exterminated by a simple explanation, usually followed by a "Why didn't I think of that?" And, finally, re Studio II and ELF articles: Continue in moderation, with the emphasis on converting them to use CHIP-8 or VIP boards. Possibly you could publish short summaries describing the conversions, hardware, etc., and provide for the purchase of the full articles at a nominal cost. - Please have a local "good guy" contact me! - John B Sewell, 11410 Braewick, Houston, TX 77035

John - Whew! Lots of good ideas and helpful hints in there! You aren't doing too badly for a self-proclaimed "novice"! Thanks and good luck. - Terry

Terry - The current controversy over whether to feature articles on the non VIP systems available to support the 1802 seems a bit foolish, in that soon most of the various "other" systems will be talking in CHIP-8. The main emphasis should be on the 1802's software ability; your angle being CHIP-8, usable by the majority in the future. As far as hardware differences, I would expect most systems installed to be pure RCA components throughout their expansions. How would you justify the articles on the Studio II? - Chuck Reed

Chuck - Surprise! Most "other" systems do not sport RCA components throughout their expansion. In fact, RCA doesn't support either of the other two "popular" 1802 systems at all. It is true that ELF and ELF II owners will be converting to VIP expansion hardware (with tedious modifications, sometimes) and using the VIP's CHIP-8, and the reason for including conversion information in the VIPER is that these converts are effectively VIP people - if they have the information they need to make the conversions. The rest of us could then benefit from their experiences and anecdotes, as they slowly work their way into the VIP fold. The argument against including ELF and ELF II information in the VIPER is that it takes away space which could be used to inform the existing VIP folks. As for the Studio II articles, we offered no justification at all for them. When RCA sold its existing stock to Radio Shack, lots of VIP people raced out and bought Studio II's at the new low price (in fact, rumor has it that there aren't any more Studio II's to be found anywhere in the country - new, that is). Then we were asked repeatedly, "How can I convert the Studio II to a microcomputer?", and given a hundred or so different reasons why that might be a neat thing to do. Eventually a reader (Jack Wright) wrote the requested article, we printed it, and we've had a lot more inquiries. (Like, "When will you have RAM?" and "When will you have cassette I/O?" and so on.) It turns out that the Studio II has a version of CHIP-8 in ROM! However. At this point, the way readers are voting, we'll probably continue the ELF conversion articles here and set up a separate information exchange for those folks interested in Studio II. Glad you asked! - Terry

VIPER - Whatever the reason for the limited popularity of the 1802, I find the device has much appeal because of its hardware simplicity: With virtually no hardware background, I was able to understand the principles of, and successfully build a microprocessor system from components. When information from VIPER, BYTE, and similar publications became available, I was able to extend my "system"s capabilities. Thus, VIPER is of value to any 1802 owner. Conversely, I think any ELF hardware enhancement or software routine is likely to be of value to VIP owners as well. Finally, I think the survival chances of a publication serving two groups with a great overlap of interests is much greater than two separate publications. VIPer purists can certainly spend the time to skip over an article of no interest to them in return for the increased chances of having anything at all. I'd like to see, perhaps as a regular feature, some detailed discussion of 1802 support chips. I reviewed the first 10 issues and found virtually nothing in this area. When I want to add new capabilities to my ELF, I'd like to know what RCA has available. I would think RCA could see their way clear to having spec sheets and application notes re-printed. Ordering information on 1802 literature would be of interest, as well as mail order sources for chips. (Jade, at least, seems to stock a variety of 1802 chips, but I don't know how comprehensive their supply is.) So keep up the good work. Enclosed is my Renewal. - Doug Smith

Doug - I think you expressed our sentiment very well. Thanks. Terry

SAVING AND RESTORING CHIP-8 VARIABLES - A MODIFICATION

by John Bennett, 3 Grafton St., Manchester MD 21102
(301) 239-3143

In volume 1, issue 10; I presented a CHIP-8 modification which allowed the user to save/restore a range of variables VX-VY. Since that modification required two CHIP-8 instructions to implement, current CHIP-8 programs could not easily be converted to execute with the modified interpreter. Since then, I have solved that problem and the modification described in this article provides the same flexibility with a single instruction.

The following modification replaces the CHIP-8 FX55/FX65 instructions with the FXY0/FXY1 instructions. The new instructions operate as follows:

FXY0 - Store the contents of 1 to 16 variables to successive memory locations beginning with the memory address pointer (I). The variables saved will include VX through VY. The memory address pointer (I) will have been incremented by the number of variables saved: (VY-VX) + 1.

FXY1 - Load the contents of 1 to 16 variables from successive memory locations beginning with the location specified by the memory address pointer (I). The varables restored will include VX through VY. The memory address pointer (I) will have been incremented by the number of variables saved: (VY-VX) + 1.

note: VY must be greater than or equal to VX

The modification takes advantage of two CHIP-8 characteristics:

1. All instructions except those whose first digit (the opcode) is 0 are processed by the initial fetch/decode routine as if they specify both VX and VY. The VX pointer is loaded into R6 and the VY pointer is loaded into R7.

2. No instructions with the F opcode currently end in either 0 or 1.

The modified interpreter begins final decoding of all F opcode instructions at location 0100. If the FXF0 instruction is encountered, a branch is made to location 015A. If the FXY1 instruction is encountered, a branch is made to location 0167. All other F opcode instructions execute as they did in the unmodified interpreter.

To install the modification, apply the following changes to CHIP-8 at the specified location:

Address	Code	Comments
006F	00	Begin final F opcode decoding at address 0100
0100	05	Load via R5 (no advance)
01	FA	And immediate with 0F
02	0F	
03	32	Branch to 015A if D=0 (FXF0)
04	5A	
05	30	Branch to 0154
06	54	
0154	FD	Subtract 1
55	01	
56	32	Branch to 0167 if D=0 (FXF1)
57	67	
58	45	Load via R5 (with advance)
59	A3	
015A	22	Decrement stack pointer
5B	87	Get R7.0 (VY pointer)
5C	52	Store in stack
5D	06	Load via R6 (VX pointer)
5E	5A	Store via RA (I pointer)
5F	86	Get R6.0 (VX pointer)
60	F3	XOR with top of stack
61	16	Increment R6
62	1A	Increment RA
63	3A	Branch to 015D if D ≠ 0
64	5D	
65	30	Branch to 0172
66	72	
67	22	Decrement stack pointer
68	87	Get R7.0 (VY pointer)
69	52	Store in stack
6A	0A	Load via RA (I pointer)
6B	56	Store via R6 (VX pointer)
6C	86	Get R6.0 (VX pointer)
6D	F3	XOR with top of stack
6E	16	Increment R6
6F	1A	Increment RA
70	3A	Branch to 016A if D ≠ 0
71	6A	
72	45	Load via R5 (with advance)
73	12	Increment stack pointer
74	D4	Return to 0042

This modification leaves locations 01F2 through 01FB available for use for other modifications.

I/O PORT DRIVER ROUTINE

by James Barnes

Tired of playing games? Put your VIP to work, using this program to monitor and control the parallel input and output ports. It uses the CHIP-8 language, modified for I/O control. Upon running the program, "IN=" and "OUT=" will be displayed on the monitor. It then reads the input port and displays the initial value of the input in hexadecimal. The output port is initially cleared to #00. Any subsequent change in the input data will be displayed and any key pressed will be shifted into the lower digit of the output port while the former lower digit replaces the upper and the resulting data is shown.

I've added two I/O instructions to the CHIP-8 interpreter in machine language, as shown in Listing 1. After this code is entered, the CHIP-8 command 'FXF2' will transfer the input data to VX and 'FXF5' will transfer VX to the output port. Since this code is located in one of the unused areas of CHIP-8 no instructions are eliminated and the modified interpreter is still fully compatible with all other programs. Listing 1 also includes a machine language subroutine at #00FC which will turn off the 1861 video controller chip. Although that routine is not used here, it should come in handy for other programs. Two of RCA's "undocumented" CHIP-8 instructions are used -- '8XYE', meaning VY*2/VX and '8XY6' meaning VY/2/VX.

The I/O Port Driver is shown in Listing 2. After displaying "IN=" and "OUT=" and initializing some variables, the main program is entered at MLOOP.

First, the input port is read. If the input data is unchanged, the data display routine is skipped to save execution time. If the input has changed, the old date is erased and the new data shown, in hex, following "IN=".

Then the program decrements the value of TKY, tests for underflow and checks whether the keypad key corresponding to TKY is pressed. If not, the output routine is skipped by a jump to TOLOOP. If a key is pressed, the previous output number is shifted left 4 bits and the new key is sent to the output port and displayed after "OUT=". The VIP's tone is sounded while the key is pressed and the program waits until the key is released. This helps prevent unwanted key inputs.

At TOLOOP, the program, as written, simply jumps back to MLOOP and the whole thing starts again. Additional computations, however, could start there (at address 026E) and end with instruction '1234' to go back to MLOOP. For instance, one could convert and display decimal equivalents of the I/O data, test and display flag 4, or whatever.

SHOBYT is a simple CHIP-8 subroutine which displays the byte in SHOW in hexadecimal format. First, the lower nibble of SHOW is displayed starting at a location six bits to the right of the current X. Then, X is reset to its original value, SHOW is divided by 16 to bring its upper nibble into the low four bits and SHOW is displayed again to show the upper nibble. The subroutine then returns.

Patterns for the labels "IN=" and "OUT=" follow SHOBYT. They are each 5 bytes high and 16 bits long.

In the listing, variable assignments follow. Throughout this explanation, I have used symbolic names for variables and program locations. I highly recommend this thinking procedure to ANY programmer, even if he uses languages such as CHIP-8 or BASIC in which symbolics are not permitted. Using such descriptive names, the programmer knows, as he is designing, what a sub-program does or what a variable contains. He does not have to waste effort figuring out addresses or assigning registers until he finally gets down to coding -- the next-to-last step in program design (just before debugging). Consider! Which is more clear? OLDIN or V8? Call SHOBYT or do subroutine at 0280?

OK, what can you do with this thing? Using very minor interconnecting hardware, you could drive any compatible digital circuit with the output port and display the circuit's response via the input port. Thus, the program turns the VIP into a basic IC or circuit card tester. An A/D and D/A pair could also be read and written by this program, giving analog voltage I/O to the VIP. Using the same programming concepts, plus some special computations, the VIP can be made to simulate any other digital circuit with eight or less inputs and outputs.

In my own case, I have connected the VIP port to a solderless breadboard using a flat cable. I can now quickly breadboard any small digital circuit and by writing a little bit of software, have the VIP control and test it for me.

Personally, I think it's about time we put our toys to work.

LISTING 1

```
!M
0000 ;                  0001 ..CHIP-8 I/O PATCH, VERS. 1.02
0000 ;                  0002 ..
0000 ;                  0003 ..SUBROUTINE DSPOFF TURNS VDC OFF
0000 ;                  0004 ..
0000 ;                  0005 ORG #00FC
00FC 2261;              0006 DSPOFF: DEC 2; OUT 1     ..VDC OFF
00FE D4;                0007         SEP 4            ..AND RETURN
00FF ;                  0008 ..
00FF ;                  0009 ..I/O EXTENSIONS TO FXAA COMMAND.
00FF ;                  0010 ..
00FF ;                  0011 ORG #01F2                ..IN EXCESS AREA 2
01F2 ;                  0012 ..FXF2 IS PARALLEL INPUT 3 -> VX
01F2 E6;                0013 FXF2:   SEX 6            ..X=VX
01F3 6B;                0014         INP 3            ..INPUT PORT
01F4 D4;                0015         SEP 4
01F5 ;                  0016 ..FXF5 IS VX -> PARALLEL OUTPUT 3
01F5 E6;                0017 FXF5:   SEX 6            ..X=VX
01F6 6326;              0018         OUT 3; DEC 6     ..OUTPUT, RESET R6
01F8 D4;                0019         SEP 4
01F9 ;                  0020 END
0000
```

```
ADRS     LABEL    CODE     COMMENT
                  ..I/O DRIVER, VERSION 1.02
                  ..DISPLAYS INPUT PORT, SHIFTS KEYS
                  .. TO OUTPUT PORT, TONE ON KEY IN
                  ..
                  ..SETUP PROCEDURE
                  ..SHOW "IN="
0200              6308      ..X=LOCN('IN=')
0202              6408      ..Y=LOCN('IN=')
0204              A2A0      ..I=PTRN1('IN=')
0206              D345      ..SHOW PTRN
0208              7308      ..X+8
020A              A2A5      ..I=PTRN2('IN=')
020C              D345      ..SHOW PTRN
                  ..SHOW "OUT="
020E              6308      ..X=LOCN('OUT=')
0210              6410      ..Y=LOCN('OUT=')
0212              A2AA      ..I=PTRN1('OUT=')
0214              D345      ..SHOW PTRN
0216              7308      ..X+8
0218              A2AF      ..I=PTRN2('OUT=')
021A              D345      ..SHOW PTRN
                  ..CLEAR VARIABLES
021C              6600      ..CLEAR KOUT
021E              F6F5      ..CLEAR OUT PORT
0220              8A60      ..KOUT -> SHOW
0222              631C      ..X=LOCN(IN)=LOCN(OUT)
0224              6410      ..Y=LOCN(OUT)
0226              2280      ..CALL SHOBYT --- INITIAL OUT
0228              F7F2      ..IN PORT -> VIN
022A              8870      ..VIN -> OLDIN
022C              6408      ..Y=LOCN(IN)
022E              8A80      ..OLDIN -> SHOW
0230              2280      ..CALL SHOBYT --- INITIAL IN
0232              6510      ..#10 -> TKY
                  ..
                  ..MAIN LOOP
                  ..FIRST, CHECK INPUT.  REPLACE
                  .. IF CHANGED.
0234     MLOOP:   F7F2      ..INPUT -> VIN
0236              9780      ..SKIP IF VIN<>OLDIN
0238              1246      .. ELSE, GO TO TSTOUT
023A              6408      ..Y=LOCN(IN)
023C              8A80      ..OLDIN -> SHOW
023E              2280      ..CALL SHOBYT TO ERASE
0240              8870      ..VIN -> OLDIN
0242              8A80      ..OLDIN -> SHOW
0244              2280      ..CALL SHOBYT --- NEW INPUT
                  ..TEST FOR KEY UNDERFLOW, DEC KEY
0246     TSTOUT:  4500      ..SKIP IF TKY<>0
0248              6510      .. ELSE, #10->TKY
024A              75FF      ..TKY-1 -> TKY
```

```
                    ..TEST INPUT KEY, SKIP IF NONE
024C                E59E    ..SKIP IF TKY=KEY
024E                126E    .. ELSE, GO TO TOLOOP
0250                6410    ..Y=LOCN(OUT)
0252                8A60    ..KOUT -> SHOW
0254                866E    ..KOUT*2
0256                866E    ..     *4
0258                866E    ..     *8
025A                866E    ..     *16
025C                8651    ..KOUT.OR.TKY->KOUT
025E                2280    ..CALL SHOBYT -- ERASE OLD KOUT
0260                F6F5    ..KOUT -> OUT PORT
0262                8A60    ..KOUT -> SHOW
0264                2280    ..CALL SHOBYT -- SHOW NEW KOUT
                    ..WAIT TILL KEY RELEASED.
0266                6F04    ..#04 -> VF
0268    KYWT:       FF18    ..VF -> TONE
026A                E5A1    ..SKIP IF TKY<>KEY
026C                1268    .. ELSE, LOOP TILL RELEASE
026E    TOLOOP:     1234    ..LOOP TO MLOOP
                    ..EXTRA FUNCTIONS START AT TOLOOP
                    ..END WITH "1234" -- GOTO MLOOP
                    ..
                    ..SHOBYT SUBROUTINE DSPLYS 2
                    .. HEX DIGITS IN SHOW AT X,Y
0280    SHOBYT:     7306    ..X+6
0282                FA29    ..PTRN(SHOW) -> I
0284                D345    ..SHOW LSD OF BYTE
0286                73FA    ..X-6
0288                8AA6    ..SHOW/2
028A                8AA6    ..     /4
028C                8AA6    ..     /8
028E                8AA6    ..     /16
0290                FA29    ..PTRN(SHOW) -> I
0292                D345    ..SHOW MSD OF BYTE
0294                00EE    ..RETURN
                    ..PATTERNS FOR 'IN='
02A0                72 23 22 22 72
02A5                20 23 A0 63 20
                    ..PATTERNS FOR 'OUT='
02AA                F4 94 94 94 F7
02AF                B8 93 90 93 90
                    ..VARIABLE ASSIGNMENTS
                    ..X=V3
                    ..Y=V4
                    ..TKY=V5
                    ..KOUT=V6
                    ..VIN=V7
                    ..OLDIN=V8
                    ..SHOW=VA
                    ..VF USED -- TEMP STORAGE FOR TONE
            END
```

DO THE SHUFFLE

by Tom Swan

Have no fear of the title, disco has not come to VIP programming -- at least by me. Though it's true I've thought V-I-P GEES has a nice ring, and an electronic Saturday Night Current Drain would bring new meaning to the use of the floppy disco, this article unleashes a different sort of Shuffle. Feet... be still!

Generating a random deck of cards is a subject well suited - but not very well explored - to both the art of game programming and the art of programming in general. In the VIP manual, the game "Acey Ducey" solves the problem of generating a random deck of cards by ignoring the suits. As only three cards are ever on the screen at a time, even if all three cards are the same, they could actually have been dealt from a real deck. The deck does not exist in memory, and a single random number generator subroutine may be used to produce each card without fear of over duplication; for instance, generating 5 aces - though if I were at the poker table, I wouldn't complain.

Poker is a good example of why this technique will fall short without trying very hard. In fact, any card game that uses more than four cards during play must contain some method of keeping track of cards that have been used, while allowing the entire deck of 52 cards in four suits to be accessed.

One possibility would be to 'remember' which cards are generated, then compare each new generation with the list of old ones, throwing out any matches until a new, unused card is finally generated. The 52nd card would take considerably longer to generate than the first, and in theory may never be generated, since random number makers are not always under command to produce all numbers in a sequence. They will, however. Given the speed of a computer, you may count on it.

That approach would probably be the most difficult to manage, though it would not be particularly difficult to program, especially if a machine language subroutine did the work of comparing.

If we throw out random number generation to produce cards, however, then we are left with the necessity of pre-programming the deck into memory. Only the selection of each card needs to be randomly produced. The cards themselves simply exist in a table that is used by the program. Each card is randomly 'looked up' in the table as needed.

Two problems remain unsolved by this approach. The program must still 'remember' all the cards that have been previously dealth, and the 52nd card will still take longer to find, as the program will presumably spend most of its time rejecting used cards while it is trying to find the last one available in the deck.

Also, each card in the table would have to me marked in some way after it is used, making resetting of the deck difficult to program (and increasing the potential for bugs to a level somewhere in the neighborhood of 'absolutely for sure').

Solving these two limitations requires a look at how a real deck of cards is shuffled and dealt. The deck never changes - only the position of each card is assumed to be randomly different following the shuffle. Each card is as easy to produce as the last, as the next card always comes from the top of the deck.

Cards do not need to be marked or remembered after being dealt. Used cards are no longer available in the deck, a fact almost to obvious to mention.

When shuffling a deck of cards, what is it you're actually doing? First you break the deck into two more-or-less equal parts; then, by using a manipulation most of us learn as children, cause one half of the deck to merge with the other. Repeating the process produces a randomly-mixed deck. The randomness is only a factor of your imperfect dexterity - were you able to shuffle perfectly each time, the same end result could be produced, given a known starting order in the deck.

The breaking of the deck into two parts seems unnecessary in a computer. We only do it because we possess two hands; it makes the two parts of the deck easier for human hands to handle.

But the actual shuffle is what intrigued me. Thinking of the action, it revealed itself to be a simple process of continually exchanging two randomly picked cards a number of times, until the deck's order becomes hopelessly (or hopefully) mixed up. When shuffling, we are actually trying to produce a random deck by repeatedly taking two cards and switching their order around. Speed and the lumpiness of the thumb are responsible for the mix - nothing else.

Generating a random deck of cards in a computer may be treated in the exact same manner with a resulting deck or array of randomly mixed bytes (representing the cards) that are childishly simple to keep track of. Two pointers are first set at two different random positions in the deck. The bytes addressed by the two pointers are then exchanged. The process is then repeated a number of times - could be a random number of times, but doesn't have to be - and the shuffled deck is ready to use.

Gaining access to the next card in the deck only requires remembering the position of the last card; not what that card was. A simple variable set as an index that can be added to a memory pointer will find the top of the deck whenever it is needed in the program. Even cutting the deck becomes possible - the other possibilities suggested would not have allowed it. Provided you include a test to reset the index to 00 after it goes beyond the last card in the deck, the index could be set to start at a random (or player selected) starting point in the deck of cards, simulating a 'cut'. While not an essential feature, it does add realism and added randomness to the play.

An even greater benefit is the fact that the deck never changes. As in a real deck, only the order of the cards becomes confused, allowing the deck to stay like it is for new games, which would in turn reshuffle the deck. Never does the deck have to be reset for its next use. All the cards are always available.

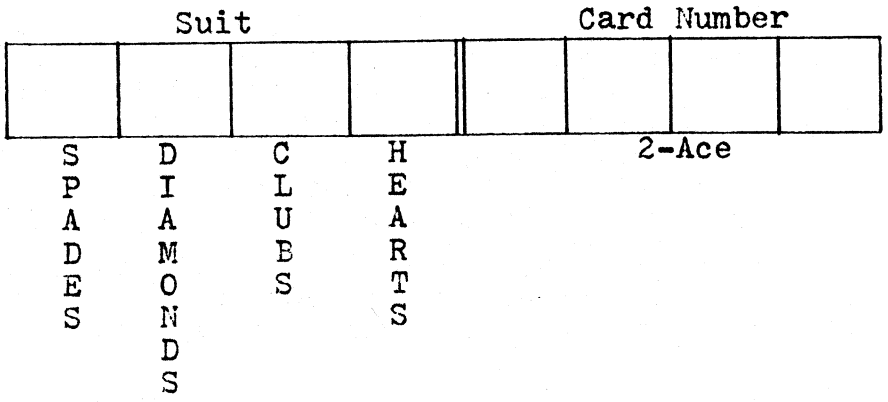
FORMATTING A CARD

Deciding how to represent playing cards as bytes in a computer memory is next. Realizing that there are only 52 cards, we know they could be numbered in sequence from 1 to 52, decimal. Deciphering which card is which, though, becomes a chore, and would involve numerous and messy greater-than or less-than and/or equal-to tests to complicate your program.

Using one byte to represent the suits from 1 to 4, and another byte for the cards from 2 to ace is a possibility that at first seems easy enough to program. But then the shuffling becomes overcomplicated for an 8-bit computer, which must now transpose two bytes for each card during the shuffle instead of one, being careful to keep each paired byte together.

If one card is represented by one byte, and is thus easier for the computer to handle, a convenient method of separating the suit from the card type must be devised. A diagram will demomstrate one possibility:

ONE 8-BIT MEMORY BYTE



In the diagram, notice the first four bits, used to designate the suit of the card. The second four bits are used for the card value from 2 to ace. The values range from 2-14 (02-0E hex), with 02=2; 0D=king; 0E=ace.

As there are four bits and four suits, it is easiest to assign a bit position in the firs ½ or nybble of the byte for each suit, as shown in the diagram. If the first bit is equal to'1', then the card is a spade; if bit #2 is '1', then it is a diamond, and so on. This provides a handy alternative to numbering the suits in binary, or hex from 01-04.

In hex, these suits and bit positions correspond to the following table:

1N = hearts
2N = clubs
4N = diamonds
8N = spades

In each entry "N" is a number from 02-0E (02-14 decimal) representing each of the 13 possible card types. (See the Deck of Cards array at the end of the program listing to examine all 52 cards.) As the deck is pre-programmed in a memory array, no tests are required to make sure that bytes fall within certain ranges. The cards exist in that table -- illegal values are impossible to obtain, assuming of course your program operates soundly.

For an 8-bit computer such as the VIP, and especially with a language like Chip-8 that privides 8 bit variables, we need a way to program the separation of suits and card types to allow their subsequent display on a TV screen or a printer. This is easily accomplished by suing the logical "AND" function to split the eight-bit byte into its two four-bit parts.

First a card is dealt from the top of the deck using the indexed pointer as previously described. The card, or other memory holding area where it may be repeatedly tested and worked on.

To obtain the card type, perform a logical "AND" on that byte using the hex value "0F" a process called masking. This will strip off the first four bits of the byte-- which represent the suit -- leaving only the card type. Performing a logical "AND", remember, results in an answer where binary ones exist only in the same bit positions that binary ones existed in <u>both</u> operands. (Only 1 "AND" 1 = 1)

To obtain the suit only, the logical "AND" may also be used but this time with the hex value "F0" as the second operand. This will strip off the card type leaving lnly the suit. To demonstrate:

GET TYPE		GET SUIT	
card	0100 0110 = 46 hex	card	0100 0110 = 46 hex
"AND"	<u>0000 1111</u> = 0F hex	"AND"	<u>1111 0000</u> = F0 hex
result	0000 0110 = 06 hex	result	0100 0000 = 40 hex

We now have two hex numbers from the one byte representing the card -- 40, which represents the suit diamonds and 06 specifying the card type or number. Thus the byte 46 is decoded into two parts, the 6 of diamonds. Each of the two results would presumably be tested to enable your program to display the correct patterns for that card on the screens.

By no means is the format just described the best or only possibility. Other masking techniques may be used to decode bytes constructed to represent different things when bits are set or not set. The example here is practical for

a language such as Chip-8, or any language that allows for testing if such and such a number is equal to some known constant. If you are programming in machine language, different formats might be devised to take advantage of your microprocessor's shifing instructions, etc. Taking your language's capabilities into consideration will make the going that much easier.

You should see how checkerboards, adventure type game boards and others may be represented in similar manner. As I did with my problem of the card shuffle, take a moment to consider how the real world operates before you try to translate it into binary simulation. The answers you look for are most probably staring you in the face. Have fun!

SHUFFLE SUB

The format for the Chip-8 subroutine below is a little different from that which usually appears in the pages of VIPER. It conforms with the format required by my program Chip-8 Assembler-3, and I realize I may be jumping the gun a little as at the time this article was written, the Assembler's availability had not yet been announced.

Those of you who do not have a copy of Assembler-3, may simply enter the subroutine as listed using the normal ROM system monitor or other Chip-8 Editor. If you are familiar with the format, however, you may want to use Text Editor-21 from Pips for Vips to prepare a copy for a relocatable source listing that may be assembled anywhere in memory. Be careful to include <u>all</u> the semicolons before the comment lines as printed here, and to follow the instructions carefully for Chip-8 Assembler-3 to insure a working copy.

In either case, the machinge language subroutine and deck of cards should be entered in by hand using the monitor. The MLS and deck array may be located on any page beginning at OMB6 and continuing to OMFF without revisions. If you do change the location, however, the Chip-8 instructions at 0506, 0508, and 051E would need to be adjusted to reflect the changes. If you want to relocate the Chip-8 Subroutine also, be careful to alter all the jumps and subroutine calls as well.

After loading in the subroutine, the sample test program at 0200 may be entered to give a demonstration of how the shuffle operates. Use the normal Chip-8 Interpreter to run the program. What you will see is a group of bytes at the bottom of your screen being acted upon by the Shuffle routine. This group is the deck that you entered at 05CC, and the activity is of course the shuffle. The Deal Subroutine is not demonstrated or used by the test program.

To use the Shuffle Subroutine is your game programming, you only need to program a subroutine call with the instruction, 2500; SHUFF, to begin shuffling. The deck will continue to be shuffled until Key 0 is pressed though you may change the byte at 050B to activate any key you want for stopping the

action. While shuffling, the bytes are not normally shown on the screen -- this was done by the test routine only to show you what was happening.

The Deal Subroutine requires that the index variable VE be initialized to 00 before the first call to the routine. After that, VE must not be changed unless its value is somehow preserved then later restored before the next call. Calling Deal with the instruction 251E; DEAL, will cause the next card in the deck to be placed in V0. The value in V0 may then be tested as described earlier, splitting the suit and the card type for further decoding to display the right patterns on the screen.

If you wish, VE may be set to any initial value from 00-33 hex thus enabling a player to "cut" the deck. The deck itself will not change, but the index VE will cause "I" to start choosing cards from the middle of the deck which is actually the same thing as a cut. All 52 cards would still be available as VE is automatically reset to 00 after the last card in the array @ 05FF has been used.

No provision for preventing a deal past the 52nd card is provided, but VE could be tested after the first call to Deal (provided the deck was not cut) to see if it equals 00. When it does, the deck is empty.

The variables V0 and V1, set to random numbers from 00-33 hex by the Gener Sub, are used by the machine language subroutine XCHNG to set two pointers, RE and RF, to random locations into the deck of cards. The two bytes addressed by RE and RF are then interchanged and the process is repeated. (It is possible that RE will equal RF at times resulting in a wasted loop -- the same card is exchanged with itself. This has no consequence on the operation of the Shuffle, however.)

During the operation of the Deal Subroutine, the "I" pointer is forst set to the top of the deck of cards at 05CC. Then the value of VE is added to "I" in order to point to the next call, the next card after the last will be set into V0.

Take care that you know which variables will be changed by the subroutine before you use them as noted in the subroutine descriptions, and remember that you do not have to reset the deck of cards in any way before the next shuffle. The cards never change, only their positions in the deck will be altered.

If you find a use for the Shuffle in a game program, I'm sure other Vipers would enjoy seeing it in a future VIPER. I know I would, but please don't make it too hard to beat. If all the money I've already lost to my VIP were real, I would have had to pack up my burro and move to the hills (el campo) a long time ago! Good luck to you.

SUBROUTINE DESCRIPTIONS

Shuffle Subroutine
- INPUT: None
- OUTPUT: Deck of cards @ 05CC shuffled until Key 0 is pressed
- CHANGES: V0 V1 "I" pointer
- CALLS: Gener at 0512; MLS XCHNG at 05B6
- LANGUAGE: Chip-8 - normal version
- TO USE: Call with 2500 instruction

Gener Subroutine
- INPUT: None (used by Shuffle Subroutine)
- OUTPUT: V0 = random number 00-33
- CHANGES: V0 VF
- CALLS: No other subroutines
- LANGUAGE: Chip-8 - normal version
- TO USE: Call with 2512 instruction

Deal Subroutine
- INPUT: VE preset to 00 before first call (VE=00-33 will cut the deck)
- OUTPUT: V0 = next card in the deck (see text for format)
- CHANGES: V0 VE (+01)
- CALLS: No other subroutines
- LANGUAGE: Chip-8 - normal version
- TO USE: Call with 251E instruction

SHUFFLE

```
0500  SHUFF :2512  GENER -- Do sub to generate random index
  02         8100 ;V1=V0 -- Preserve V0 in V1=Index #2
  04         2512  GENER -- Do sub to generate random index
  06         A5CC ;CARDS -- Set "I" to deck of cards
  08         05B6 ;XCHNG -- Do MLS, Exchange two cards
  0A         6000 ;V0=00 -- Let V0=00 for key press test
  0C         E09E ;SK=KY -- Skip if Key 0 is pressed
  0E         1500  SHUFF -- Else continue shuffle
0510         00EE ;RET   -- Return to caller
```

GENER

```
0512  GENER :C03F ;RND   -- V0=random number , 00-3F
  14         6F33 ;VF=33 -- Let VF=33 (the 52nd card)
  16         8F05 ;VF-V0 -- Subtract VF-the random number
  18         3F01 ;SK=-  -- Skip if V0≤33
  1A         1512  GENER -- Else regenerate new number
  1C         00EE ;RET   -- Then return to caller
```

DEAL

```
051E  DEAL  :A5CC ;DECK  -- Set "I" to deck of cards
  20         FE1E ;I+VE  -- Add value of VE index to "I"
  22         F055 ;GET   -- Let V0=byte (card) addressed by "I"
  24         7E01 ;VE+01 -- Add 01 to VE index for next call
  26         4E34 ;SK≠34 -- Skip if VE not past last card
  28         6E00 ;VE=00 -- Else reset VE=00
  2A         00EE ;RET   -- Then return to caller
```

XCHNG- MACHINE LANGUAGE SUBROUTINE

```
05B6  22  DEC   R2 ;Decrement stack pointer to free location
  B7  9A  GHI   RA ;Get RA.1 (High part of "I" address)
  B8  BE  PHI   RE ;Put in RE.1 - Pointer A
  B9  BF  PHI   RF ;Put in RF.1 - Pointer B
  BA  8A  GLO   RA ;Get RA.0 (Low part of "I" address)
  BB  52  STR   R2 ;Push onto stack
  BC  F8  LDI      ;Load "F0" byte into D register
  BD  F0
  BE  A6  PLO   R6 ;Put in R6.0 to address Chip-8's V0
                    variable
  BF  46  LDA   R6 ;Get value of V0, advance R6 to V1

05C0  F4  ADD      ;Add to byte on stack forming random index
  C1  AE  PLO   RE ;Put in RE.0 - Pointer A
  C2  06  LDN   R6 ;Get value of V1
  C3  F4  ADD      ;Add to byte on stack forming random index
  C4  AF  PLO   RF ;Put in RF.0 - Pointer B
  C5  0E  LDN   RE ;Get byte (card #1) at pointer A
  C6  52  STR   R2 ;Push onto stack to store temporarily
  C7  0F  LDN   RF ;Get byte (card #2) at Pointer B
  C8  5E  STR   RE ;Store card #2 at old card #1 position
  C9  72  LDXA     ;Pop card #1 resetting stack pointer
  CA  5F  STR   RF ;Store card #1 at old card #2 position
  CB  D4  SEP   R4 ;Return control to Chip-8 Interpreter
```

DECK OF CARDS

```
05CC  12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E
05D9  22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E
05E6  42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E
05F3  82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E
```

TEST PROGRAM

```
0200  BEGIN :0206  MLS   -- Do machine language sub @ 0206
  02         2500  SHUFF -- Do Shuffle Subroutine
  04  STOP   :1204  STOP  -- On return (Key 0), halt program

  06  MLS    :F805 ;CHNGE -- Set display page to 0500 to
             BBD4 ;DISP  -- View action of Shuffle Subroutine
```

SIMPLE SIMON GAME

by Pete Kellner

This program simulates the electronic Simple Simon game made by Milton-Bradley. I have had only a limited access to the game itself (in a store), so I'm not sure how closely I managed to reproduce it here. The program did, however, save me from having to buy the game for my daughter.

The game uses the Simple Sound Board VP595, but doesn't require the CHIP-8X interpreter. Instead, the old CHIP-8I is used (VIPER, Volume 1, Issue 3), which adds three new commands the the CHIP-8 interpreter:

B0KK - output KK to port (Simple Sound Board)
B1X0 - output contents of VX to port (Simple Sound Board)
B1X1 - Read input from port (not used for the Sound Board)

Note: Be sure to use the corrected CHIP-8I code. The corrections are in issue 5, volume 1 of the VIPER.

The hex keys 2, 4, 6, and 8 represent the four squares on the screen, and the object of the game is to reproduce the machine generated random sequence by pressing the appropriate keys in the correct sequence, until an error in entry is detected.

The program then goes into error mode, exercising the Sound Board in a siren-like fashion. While in this mode, pressing the hex key F will restart the game, and pressing the hex key E will replay the last sequence.

MAIN LOOP

```
0200   22E0  Call Subroutine:  Draw Play Field
0202   6700  V7=0              Initialize counter
0204   7701  V7=V7+1           Increment counter
0206   6E00  VE=0              X, Y location
0208   F729  I=V7              (LSDP)
020A   DEE5  Show 5MI @ VE, VE
020C   C003  V0=RND
020E   A400  I=400             Storage of RND sequence
0210   F71E  I=I+V7
0212   F055  MI=V0             Store the random number
0214   2320  Call Subroutine:  Playback
0216   6E00  VE=0
0218   8870  V8=V7
021A   228E  Call Subroutine:  Keypress
021C   2240  Cal Subroutine:   Generate Display and sound
021E   A400  I=400             Storage of RND sequence
```

```
0220   7E01  VE=VE+1
0222   FE1E  I=I+VE
0224   F065  V0=MI                Get the random number
0226   50C0  If (V0=VC), skip.    Correct response
0228   22C0  Call Subroutine:     Error handler
022A   78FF  V8=V8-1
022C   3800  If (V8=0),skip.      End of loop
022E   121A  Goto 021A

0230   6E00  VE=0
0232   F729  I=V7                 (LSDP)
0234   DEE5  Digit display off
0236   2310  Call Subroutine:     Timing
0238   1204  Goto 0204            Next iteration.


              DISPLAY AND SOUND GENERATION SUBROUTINE

0240   4000  If (V0≠ 0), skip.
0242   1266  Goto 0266
0244   4001  If (V0≠ 1), skip
0246   125E  Goto 025E
0248   4002  If (V0≠ 2), skip
024A   1256  Goto 0256
024C   6F00  VF=0
024E   6A18  VA=18                X Coordinate

0250   6B10  VB=10                Y Coordinate
0252   6D6F  VD=6F                Tone value
0254   126C  Goto 026C
0256   6A20  VA=20                X Coordinate
0258   6B08  VB=8                 Y Coordinate
025A   6D8B  VD=8B                Tone value
025C   126C  Goto 026C
025E   6A10  VA=10                X Coordinate

0260   6B08  VB=8                 Y Coordinate
0262   6DA6  VD=A6                Tone value
0264   126C  Goto 026C
0266   6A18  VA=18                X Coordinate
0268   6B00  VB=0                 Y Coordinate
026A   6DD2  VD=D2                Tone Value
026C   A308  I=0308
026E   DAB8  Show 8MI @ VA,VB     Square of light

0270   B1D0  VD - Tone board
0272   632F  V3=2F                Timing constant
0274   8170  V1=V7
0276   811E  Shift left.  V1=V1*2     This arithmetic increases
0278   811E  Shift left.  V1=V1*2     the speed for every iter-
027A   8315  V3=V3-V1                  ation
027C   F315  Timer=V3
027E   F318  Tone duration=V3

0280   F407  Timer=V4
0282   3400  If (V4=0), skip
0284   1280  Goto 0280
0286   DAB8  Display off          (Square of light)
0288   00EE  Return
```

KEYPRESS SUBROUTINE

```
028E   6405  V4=5

0290   6002  V0=2
0292   6104  V1=4
0294   6206  V2=6
0286   6308  V3=8
0288   E0A1  If (V0≠ hex key), skip
029A   6400  V4=0
029C   E1A1  If (V0≠ hex key), skip
029E   6401  V4=1

02A0   E2A1  If (V2≠ hex key), skip
02A2   6402  V4=2
02A4   E3A1  If (V3≠ hex key), skip
02A6   6403  V4=3
02A8   4405  If V4=05), skip
02AA   1298  Goto 0298          Next key
02AC   8040  V0=V4
02AE   8C40  VC=V4

02B0   00EE  Return
```

ERROR HANDLING SUBROUTINE

```
02C0   6001  V0=1               Tone duration
02C2   61FF  V1=FF              Tone value
02C4   B110  V1 - Tone Board
02C6   F015  Timer=V0
02C8   F018  Tone udration=V0
02CA   F207  V2=current timer value
02CC   3200  If (V2=0), skip
02CE   12CA  Goto 02CA
02D0   71FF  V1=V1-1            Decrement the tone value
02D2   650F  V5=0F              * Press F to restart
02D4   E5A1  If (V5≠ hex key) skip
02D6   1200  Goto 0200          Start new game
02D8   650E  V5=0E              * Press E to play back the
02DA   E5A1  If (V5≠ hex key) skip     blown sequence
02DC   2320  Call Subroutine:   Playback
02DE   12C4  Goto 02C4
```

DRAW PLAY FIELD SUBROUTINE

```
02E0   00E0  Erase screen
02E2   A300  I=0300             Play field display
02E4   6118  V1=18
02E6   6210  V2=10
02E8   D128  Show 8MI @ V1,V2
02EA   6120  V1=20
02EC   6208  V2=08
02EE   D128  Show 8MI @ V1,V2
```

```
02F0   6110  V1=10
02F2   6208  V2=08
02F4   D128  Show 8MI @ V1,V2
02F6   6118  V1=18
02F8   6200  V2=00
02FA   D128  Show 8MI @ V1,V2
02FC   2310  Call Subroutine:  Timing
02FE   00EE  Return
```

PLAY FIELD DISPLAY DATA

```
0300   FF 81 81 81 81 81 81 FF
0308   00 00 3C 3C 3C 3C 00 00
```

TIMING SUBROUTINE

```
0310   635F  V3-5F
0312   F315  Timer=V3
0314   F307  V3=current timer value
0316   3300  If (V3=0) skip
0318   1314  Goto 0314
031A   00EE  Return
```

PLAYBACK SUBROUTINE

```
0320   6800  V8=00              Initialize sequence counter
0322   A400  I=400              Storage of RND sequence
0324   7801  V8=V8+1            Increment sequence counter
0326   F81E  I=I+V8             Set pointer to next element
0328   F065  V0=MI              Get the next element of sequence
032A   2240  Call Subroutine:   Generate Display and sound
032C   3780  If (V7=V8) skip    Test for end of sequence
032E   1322  Goto 0322

0330   00EE  Return
```

The object of the game is, of course, to obtain the longest sequence. I find it difficult to get beyond a ten note sequence, depending on how repetitive the random pattern becomes.

Other unrelated tips:

To get a very simple joystick control for my VIP, I acquired a joystick from manufacturers of electronic games such as ATARI. Any type can be used, if it has four on/off contacts in the four directions. By connecting these contacts in parallel with the hex key buttons 2, 4, 6, and 8, I have a very effective X,Y controller that works with many existing VIP games.

LITTLE LOOPS

by

Tom Swan

Machine Language Fantasies! I'll bet you have one. My favorite is designing relocatable code, then I find I have to change program counters in the middle of the stream and - POOF! - out comes the eraser and in comes another location-dependent sub routine. Is it possible, I asked, for another register to become the program counter <u>without</u> knowing the address this will happen? Yes! Here's how. R3 starts as the program counter, R4 ends as the program counter.

RELOCATABLE

```
0000  83  GLO  R3  ;R3.0→D
  01  A4  PLO  R4  ;D→R4.0
  02  24  DEC  R4  ;R4 - 01
  03  93  GHI  R3  ;R3.1→D
  04  B4  PHI  R4  ;D→R4.1
  05  D4  SEP  R4  ;PC = R4
  06
```

LOCATION DEPENDENT

```
0000  93  GHI  R3  ;R3.1→D
  01  B4  PHI  R4  ;R4.1 = R3.1
  02  F8  LDI
  03  06            ;Load start address
  04  A4  PLO  R4  ;Into R4.0
  05  D4  SEP  R4  ;PC = R4
  06
```

The results of the above two routines are identical! (Prove this for yourself using the excellent Breakpoint and Register Display Program in VIPER Oct. 1978.) As both programs are the same length, the <u>only</u> advantage to the first is that it can be located anywhere in memory, except across page boundaries. The disadvantage to the first is its confusing nature, and the fact that it repeats two instructions.

At ML 0000, the value of 01 is brought into the D register. R3 points to 0001 <u>while</u> the 83; GLO R3 is

executed. ML 0001 places this value into R4.0 which is then decremented at ML 0002. This leaves R4.0 pointing to the first line of the routine, at ML 0000. ML's 0003 and 0004 set the page address into R4.1, and ML 0005 makes R4 the program counter, leaving R3 pointing to the following line (at ML 0006). However, as R4 points to the first line of the routine (@ ML 0000) another 83; GLO R3 instruction is performed, this time with the routine running in R4. Since R3 points to ML 0006, when R3.0 (06 here) is placed in R4.0 at ML 0001, a jump to ML 0006 is caused. R4 then continues as the PC.

Houdini would have been proud! However, this may give severe indigestion to structured programming programmers. But, I set out to satisfy my relocatable code tooth, and this did it!

PROJECT: Using the relocatable PC swap, design a program that loops through a function twice, but is completely relocatable, using no branching. (Hint- You'll have to use two program counters to do this.) Answer next month.

NEXT MONTH: The Instruction With The Ridiculous Name

Please feel free to write to me at: Apartado #38, Taxco, Gro., MEXICO (Stamped Envelopes appreciated) Good luck with your programming!

ERRORS IN THE RCA COSMAC VIP USER'S GUIDE (VP-320)

by Eugene Fleming

The COSMAC VIP User's Guide is a welcome aid to learning the CHIP-8 language for many computer neophytes - like myself. However, it is not without problems, and seems to have been hastened to press to meet our need without complete checking.

The printing errors at the top left of pages 28 and 42 are not likely to confuse anyone who knows printers can make mistakes. The page labels, 6XKK for page 28, and CXKK for page 42, seem to have been interchanged.

There is a more serious problem with the program on page 36 used to illustrate the EXA1 instruction. When loaded as printed, the program would not run properly on my VIP. Data from 0300 MI could be displayed, but would only blink when any key was pressed, and would reappear after blinking once. The figure would not move about the screen at all, as the text indicates it should. After checking the code in memory several times, to be triple sure of proper keying-in, it was decided to delete line 0214. How to do that? I decided the alternatives were to replace it with line 0216 and move the rest of the program up a notch, or to replace 0214 with a do-nothing line. Since I'm not fond of re-entering, the latter course was used. Line 0214 was changed to 1216, Go To Next Line. It worked! My first debugging attempt was at least a partial success. The pattern from 0300 could be moved about the screen, though it blinks continuously. The "Wait For Key Press" on 0214 somehow inhibited the execution of the remainder of the program. It is not clear why this is the case. This allows use of the EXA1 instruction to be illustrated, which is not true of the suggested "fix" on page 21 of the April, 1979 VIPER.

With that small success as encouragement, I decided to modify the program on page 38 as illustrative of the FX29 instruction. As written, the program continuously writes the hex digits (hexits?). Most of the time there is just a scrambled mess across the top of the screen. The following changes will display the digits 0 - 9 across the top of the screen and then stop the program.

Line #	Old	New	Comment on New Code
020E	1206	313C	Skip if V1=3C
0210	--	1206	No, loop back to 0206
0212	--	1212	Yes, stop

The most confusing errors so far encountered in the manual are on page 33, in the discussion of the 3XKK instruction. The first problem is in the right column, third paragraph. There are two references to Y=1B. As the program is written, that can never happen! V2 is set to 18 by line 0204 and is decremented in steps of 06 by line 0210. Therefore, V2 will always be an even number. 1B in hex equals 27 in decimal, which is definitely an odd num-

ber. If Y=1E, the program runs beautifully, since 1E hex equals 30 decimal, which is a multiple of 06. For this reason, change to Y=1E in both text and program listing in this column.

There is much confusion in the right column of page 34. A listing is given below which seems to be the simplest workable form. At least two other workable schemes were found. The explanation of the changed lines from the suggested listings on page 34 follow the listing.

```
0200    A300    Set data pointer I=0300
0202    613A    Set V1=3A
0204    6218    Set V2=18
0206    D125    Display data from 0300
0208    D125    Display again to erase
020A    71FE    V1+FE, results in -2
020C    3100    Skip if V1=00
020E    1206    No, go to 0206
0210    72FA    Yes, V2-06
0212    613A    Reset V1=3A
0214    32FA    Skip if V2=FA
0216    1206    No, go to 0206
0218    1218    Yes, stop here.
```

Line 0204: Instead of letting Y=1B, setting it equal to 18 makes it an even number, and allows a display of up to five lines. Leaving Y=1E does not allow the whole pattern to show on the first pass across the screen.

Line 0210: 72F9 is the suggested instruction. That subtracts 07 instead of the 06 which the writer seems to have wanted to subtract. Adding FA to the previous hex number is like subtracting 06.

Line 0212: Program will run with the original instruction left at 6100, but most of the lines repeat four times unless it is changed to match line 0204. This instruction was not mentioned in the revised program.

Line 0214: The 3200 instruction suggested in text will not work in any of the "fixes" tried. It can be made to work by adjusting the starting point specified by line 0204, but even then it will not allow display of the pattern on the top line of the screen. By regressing a bit further, the top line can be displayed. Here is the formula, in hex: 18 - (5 * 06) = FA. This result assumes use of the subtraction system shown in the left column of page 34.

It is hoped that these observations and corrections will assist others and prevent their spending hours trying to locate the problems.

1Kx4 CMOS
STATIC RAMS
6514 $8.00

4Kx1 CMOS
STATIC RAMS
6504 $8.00

4K CMOS Memory IC's

Eliminate heat build-up &
Reduce Power Supply Requirements

4Kx1 6504 CMOS memory chips $8.00 each
The 6504 is plug compatible with 4044 NMOS

1Kx4 6514 CMOS memory chips $8.00 each

BOTH CHIPS FEATURE

Low Power Standby	<2.5 mW Max
Low Power Operation	<25 mW/MHz Max
Fast Access Time	<300 nsec Max

TTL Compatible Input and Output
Common Data Input-Output
Industry Standard 2114 Type Pinout
On Chip Address Register
Easy Interfacing With Multiplexed Bus uP's (8085)
ALL CHIPS TESTED 100% FUNCTIONAL

DIGITAL GROUP
Equipment Users

32K Memory Boards without memory chips $60.00
These boards are designed to use either the 6504 or 4044 NMOS memory chips

	KIT	ASSEM
32K memory boards with 16K of memory	$284	$359
32K memory boards with 32K of memory	$508	$608

ROTA-STROBES for monitoring and adjusting
PHI-DECK Tape Speed $4.50 each

Send Orders To: EMERGE SYSTEMS
P.O. Box 2518
Satellite Beach, Fl 32937

ALL ORDERS MUST BE PREPAID WITH CHECK OR M. O.
Allow time for personal checks to clear
Florida residents add 4% sales tax

1Kx4 CMOS
STATIC RAMS
6514 $8.00

4Kx1 CMOS
STATIC RAMS
6504 $8.00

Club Members:

Emerge Systems is offering the above IC's at the lowest prices ever!

These IC's normally cost several times our price.

We believe these prices will allow the hobbyist to take advantage of the benefits available with CMOS. Military and industry have used these IC's for a long time, but they have been priced beyond the budget of most hobbyist.

CMOS IC's use so little power to operate that they feel cool to the touch when running.

Although the access time for these IC's are tested to be less than 300 nsec max, they normally test closer to 150 nsec. We have hobbyist using these IC's at 4 MHz with Z80 micro's.

For the hobbyist who has a sysem which uses the TMS-4044 or other compatible memory IC's, they can be replaced with the 6504 which which is plug compatible with the TMS-4044. For those planning to add additional memory, these IC's will provide an economical alternative and take advantage of CMOS.

For those hobbyist who have contemplated building portable battery, or solar powered computers, these IC's will allow him to do so at a much lower cost and power requirements than with TMS-4044 or other NMOS IC's.

We offer discounts to clubs for quantity purchases. So, if there is sufficient interest in your club to warrant a quantity purchase please contact us at our above address, or call: 305-773-7878

EXPAND YOUR SYSTEM WITH AN ARESCO SYSTEM EXPANSION BACKPLANE

The ARESCO Backplane board allows you to plug up to five boards at once into your Studio II or VIP I/O expansion connector.

The board consists of a 22-pin edge connector with mounting holes for up to five dual 22 pin sockets.

On the Studio II this allows you to connect your PROM monitor board and still have four sockets left over for future expansion, such as additional RAM, cassette interface, ASCII keyboard interface, or external controllers.

On the VIP this means that you can plug in multiple I/O boards, rather than being limited to the single socket provided on the VIP. Like-numbered pins are bussed together on all five connector sockets. The 22 VIP I/O socket connections are connected and the 22 unused pins are available for custom connections to VIP I/O, control, address or data signals.

The board is drilled to accept five dual 22 pin sockets using 3.5 mm spacing between the two rows of pins. This includes connectors such as Amphenol 225-22221-110 or AMP 2-530668-6. Connectors using 5 mm spacing may by forcefit.

The ARESCO Backplane board is designed to mount vertically with plugin expansion boards mounted horizontally, face down. Mounting a VIP Simple Sound board on the Backplane board will cause the Simple Sound board to extend over the keypad area.

The ARESCO Backplane board will provide the foundation for addition of a variety of I/O expansion boards which ARESCO hopes to provide for the Studio II and the VIP. Even without these boards, the Backplane board will allow you to add a variety of custom interfaces to your Studio II or VIP.

ARESCO has a limited stock of inexpensive 44-pin connectors. These use 5mm spacing and require slight bending of the connector leads to fit properly. Connectors so mounted will not sit flush on the backplane board but will provide sufficiently rigid mounting for all planned expansion boards.

The ARESCO Backplane board is supplied in two forms:

1. Backplane board only - customer must supply 44-pin connectors - $20.
2. Backplane board with four connectors mounted. Customer must solder all pins - $36.

Please send me an ARESCO Backplane board type #1 ($20)____ #2 ($36)_____. I enclose a check or money order for prepaid shipment. C.O.D., MasterCharge or VISA shipments will be charged for shipping and C.O.D. fees.

PIPS FOR VIPS ... A PROGRESS REPORT

When we introduced PIPs, we were unsure of the reception it would receive. WE knew it was good. What would YOU think? From the responses we've received, it's apparent that you agree with us. At this point, we have fewer than 100 copies remaining of the original press run. Almost everyone got in on the pre-publication offer, so we were able to get it to press and out to you before the announced publication date!

One of the things that took US by surprise is the number of requests we've received for modification information - for the Text Editor 21 program, so it can be used with an ASCII keyboard. Tom Swan dug into that one right away - and he's come up with another 160 pages of material that will give you cause to turn cartwheels! Not only is there a keyboard modification so you can use the Text Editor 21 with a full ASCII keyboard, but he's included an ASSEMBLER FOR CHIP-8. The second volume of PIPS will contain a variety of different mods to the Text Editor, for use with external keyboards, and the CHIP-8 Assembler, which will allow you to write CHIP-8 code with the Editor program - but allows labels rather than absolute addresses. This means you can keep a library of your favorite CHIP-8 routines, put them all together with the Editor, and then assemble the completed programs. The VIP is looking more and more like a 'big' machine every day!

So here we are again, with a pre-publication offer for PIPS Volume 2: If we receive your order before October 15, you get the book and the cassette tape for $14.95. If we get your order after October 15, the price is $19.95. This time around, we'll deposit your checks and MC/VISA sales slips in a savings account until the day we pick it all up from the printer - some of your checks hadn't cleared by the time we had to pay for PIPS Volume 1, and it was a bit of a hassle for us. And here's even better news: If PIPS 2 is as big a success as PIPS 1, we have a third volume lurking in the wings, just waiting to be published.

As a further incentive to add PIPS to your library, we have already converted the Volume 1 PIPS programs to run on the VIP II - so you'll have a ready-to-go library available if you decide to get the VIP II when it comes out.

And, finally, we'd like to hear from PIPS purchasers who have modified Space Wars or Surround to make use of the Color, Simple Sound, or Super Sound boards. And we hope to start seeing programs in the VIPER which use the CHIP-8 Messager.

If you're willing to invest in PIPS 2, send us a check or give us your Master Charge or VISA numbers as soon as you can. We can't go to press until we have enough orders to cover the printing costs, unfortunately. If you specify COD, we're willing to ship via UPS COD - but you get to pay the $1.00 handling charge and the up-to-$2.00 shipping fee charged by UPS. Remember: PIPS 2 for $14.95 until October 15!

PRODUCT REVIEW

ATV RESEARCH MICROVERTER

by Rick Simpson

When I originally got my VIP, I used it with a video monitor, which worked just fine. But when I wanted to use the color card, I had a problem. The color card output is a video sig-signal, and my color TV requires an RF signal. I wasn't about to buy a $600+ color video monitor, so I had to attach an RF modulator to the VIP output. I didn't want to assemble and modify a Pixie Verter - and ahave to do all the work necessary to put it in a metal box, etc., so I finally attached the VIP to a Sup-R-Mod purchased from my friendly neighborhood computer store.

More problems. The Sup-R-Mod requires +12V, which my VIP doesn't have (fixed that by shorting out a resistor). The connector was wrong (tear off the video cable on the color card and wire it directly to the modulator). And the whole thing ended up with a mass of cables, switch boxes, and other little gadgets dangling her and ther (for which there is no known solution).

Now I've discovered the ATV Microverter. It's powered by four penlight batteries which should last forever (what? no digging around in my VIP?), has a neat jack for video input (but I still have to change the connector on the color board cable), and it doesn't require any connection to my color TV (no dangling boxes and gadgets). The Microverter has a half-inch stub antenna sticking out the back, which broadcasts to the UHF loop antenna on the TV. Presto! No cables! No switchboxes dangling here and there! And lest you fear that the entire neighborhood will receive you on their TV sets, I hasten to assure you: I could not pick up the transmission when I moved the Microverter more than one foot from my TV.

The only assembly required is to remove four screws and insert the four batteries, then close it back up again. And, of course, to fix that cable on the color card. The display is excellent. No trace of RF interference from the VIP itself.

In fact, I was so impressed with the ATV Microverter that we decided to stock it here at ARESCO so it would be available immediately for VIP users. The price is $34.95 - and it's well worth it. You can call us (301) 730-5186 with Master Charge or VISA orders, or UPS COD orders (you get to pay the UPS charges) or mail your order to Box 1142 Columbia MD 21044.

Editor's Note: VIPER readers who have purchased any of the RCA (or other) boards for their VIPs are encouraged - nay, URGED to write reviews. Many people are holding off from their purchases until they get input from someone who "owns one". Share your info with all of us!