VIPER

June-July 1981

Volume 3, number 2 Journal of the VIP Hobby Computer Assn.
The VIPER was founded by ARESCO, Inc. in June, 1978

****

Contents

VIPHCA INFO 3.02.01

EDITORIAL 3.02.02

ERROR TRAP 3.02.02

READER I/O 3.02.02

SOFTWARE
Little Loops: Bounce-Off, a CHIP-8 game
(by Tom Swan)..... 3.02.03

ADVERTISEMENT 3.02.10

TUTORIAL
Machine Code: a new feature series
(by Paul Piescik)..... 3.02.11

ADVERTISEMENT 3.02.16

HARDWARE
CHIP-8 PROM Board for the VIP
(by Bob Casey) 3.02.17

SOFTWARE
Boomerang Puzzle, a game in VIP Tiny BASIC
..... 3.02.18

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 which govern the operation of the Association. Mr. Sills is serving as Director of the Association, as well as editor and publisher of the VIPER.

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. Individual copies of the VIPER and past issues, where they are available, may be sent to interested people for $3 each. Annual dues to the Association, which includes six issues of the VIPER, are $12 per year.

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

Membership in the VIP Hobby Computer Association is open to all people who desire to promote and enjoy the VIP and other 1802 based computers. Send a check for $12 payable to "The VIP Hobby Computer Association" c/o Raymond C. Sills 32 Ainsworth Avenue, East Brunswick, NJ 08816 USA. People outside of U.S. Canada, and Mexico, include $6 extra for postage. All funds in U.S. dollars, please.

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. Articles, letters, programs, etc., in camera ready form on 8½ by 11 inch paper will be given preferential consideration. Many dot-matrix printers will not copy well, so this material has to be re-typed before it can be used. Please send enough information about any programs so that the readers can operate the program. Fully documented programs are best, but "memory dumps" are OK if you provide enough information to run the program. Please indicate in your material key memory locations and data sections.

Advertising Rates:

1. Non-commercial classified ads from members, 5¢ per word, minimum of 20 words. Your address or phone number is free.

2. Commercial ads and ads from non-members, 10¢ per word, minimum of 20 words. Your address or phone number is free.

3. Display ads from camera ready copy, $6/half page, $10/page.

Payment in full must accompany all ads. Rates are subject to change.

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

EDITORIAL

Dear VIPers:

VIPER 3.01 seems to have been very well received by just about everyone I was able to ask about it. Naturally, no one publication or issue can be all things to all people, but I'll try to make it interesting and informative to as many of you as possible. I think it would be nice, whenever possible, to have a "mix" of material so that there will be something for most of the membership in any one particular issue.

In this issue, we have a nice new game fresh from the programmer's pencil of Tom Swan. The game is quite simple in concept and format, but, believe me, it's a real "toughie" to play! It will be a long time before you master this one!

Also in this issue of VIPER is a new feature: a column (?) that is devoted to machine language. Paul Piescik of Cuddly Software has written a very nice article which takes us into the frequently feared world of machine language. But lest you become too concerned, this first venture is but a few short steps into the Black Forest, with Paul patiently and securely holding our hands all the while. CHIP-8 is a delightful programming language for 1802 users, but there are times when nothing but machine language will do. Once we learn that machine language is simple, logical, and above all, knowable, we will have begun to truly understand how our computer works. And that will permit us to use the real power of the computer: a tool for thinking.

If you have any questions or suggestions to offer to Paul about his machine language series, you can either send them in to me or drop a postcard directly to Paul. But bear in mind that Paul cannot respond to you directly, but he will answer questions, etc. in his column. He would very much appreciate hearing from you to provide the "feedback" necessary to keep the material interesting and useful.

Yours,

Raymond C. Sills

ERROR TRAP

Last issue's article by Bob Casey had a typo (my fault) in the code for the FXF3 demonstration program. 0214 should be D115, not D125. The program is the one in the right-hand column on p. 3.01.18.

READER I/O

Dear Mr. Sills:

Thank you for reviving VIPER. It is almost a necessity for a VIPer who has retired to northern New Hampshire.... My current interest is in interfacing a recently acquired ASR-33 for hard copy & keyboard functions. Thus far my VIP can tell the teletype to do so but I haven't had enough time to generate a "working" keyboard program. When it is complete, I will send along a copy for possible publication.

My next VIP project will be to drive 8 toy type DC motors for another hobby. Still later, interfacing a 14" CRT terminal by Control Data Corp. is planned, but the lack of service manuals on it and the ASR-33 is a handicap, to say the least. I would like to hear from others with similar interests.

W.A. Andrews, PO Box 115, Haverhill, NH 03765

BOUNCE - OFF

by Tom Swan

Well, it's good to be back.

Every once in a while I get the itch to sit down with my little blue plastic VIP buddy and whip up a good ol' Chip-8 (the regular kind) ball-bouncing video game. One of the great things about the VIP computer is how easily such games go together, a few hours work and a few hundred bytes of code and there it is! One more video "frustrater" to add to the cassette collection.

I have seen Bounce-Off (under various aliases) running on other computers, Atari and Pet in particular. The second book of VIP games published by RCA has a similar game although the playing action of Bounce-Off is different from John Fort's Deflection.

When you run the program, you will first see a question mark in the upper left corner. Press any key at this time to select the number of targets you want to deal with. (Select from three to five targets your first few times until you get used to the game.) Pressing key zero gives you 16 targets.

After this you will see the targets (single white dots) displayed on the screen. A ball will be moving horizontally across the screen. The object of the game is to direct the moving ball over the targets. Each target you hit scores one point. You know you got one when you hear a beep. So far so good. To win, all you have to do is get all the targets. Pretty simple isn't it? (Heh, heh, heh...)

Two hex pad keys control the game. To end the game and see your score (in others words, to give up) press key 1. Once you press key 1, you will have to reset the computer to restart the game so don't press this key unless you really intend to quit.

Key F is used to place a three-bit diagonal slash on the display at the current position of the moving ball. You must press and release key F for it to work. A light tap seems to work best. Holding key F down will cause the game to pause, but there is usually no good reason to do this.

When the moving ball hits a slash that you have placed on the display, the ball will change direction. I won't describe how the ball moves here. You'll have more fun working that out on your own. Experiment a little. Computing time is cheap when the computer is yours!

Be careful not to get trapped in a corner. This can happen quite easily, and once you are trapped, there is nothing to do but give up by pressing key 1. It is always possible to win the game, at least

from the start. It is easy to get yourself into several hopeless situations however, and then you lose. If you manage to score all the targets, the program will go on automatically to show you your score.

There are three parts to the score. The first number in the upper left corner is the number of targets you hit. The number to the right is the number of targets you selected at the start of the game. If these two numbers are the same, you have won. Congratulations. The third number (directly below your score) is the number of slashes you used during the game. A perfect score is to have all the numbers the same. This is extremely difficult to do when there are more than two targets, and in fact, may at times be impossible because the ball may have to change direction more than one time to line up with one of the targets. However, the objective is to use as few slashes as possible.

A few restrictions exist. A slash may never be placed over a target or over another slash. In order to place a slash on the display, there must be enough room at that position. Of course after a target is removed, you may place a slash at the old position of the target.

Enter the game as shown starting at address 0200. Watch the addresses -- there is a gap from 02E0 to 02FF. Use a copy of the Chip-8 interpreter as supplied with your computer. This goes in locations 0000 to 01FF (load two pages from another game if you don't have the interpreter stored separately).

Good luck and happy bouncing!

BOUNCE - OFF by Tom Swan

VARIABLES

V0 - Scratch
V1 - "
V2 - "
V3 - "
V4 - Score (# targets hit)
V5 - Score (# slashes used)
V6 -
V7 -
V8 -
V9 - Number targets selected
VA - X direction; -1 left; +1 right
VB - Y direction; -1 up ; +1 down
VC - X Ball and target display
VD - Y Ball and target display
VE - Scratch
VF - Flag (for hits & 8XYN commands)

MEMORY MAP

0000 - 01FF -- CHIP-8 interpreter (You supply this code.)
0200 - 02DE -- Main program routines
0300 - 03E0 -- Sub routines

```
03AA - 03CB     -- Constants and display patterns
0400 - 041F     -- Program work area.  Don't put modifications here!
```

THE PROGRAM

```
0200    START:  A3AA    ;POINT TO INITIAL VARIABLE VALUES
  02            FF65    ;INITIALIZE ALL VARIABLES
  04            A3BE    ;POINT TO '?' MARK
  06            D128    ;SHOW IT @ V1,V2
  08            F90A    ;GET # TARGETS DESIRED (1..15)
  0A            4900    ;SKIP NEXT IF V9 <> 0
  0C            6910    ;MAKE 0 INTO 16 FOR (1..16) RANGE
  0E            00E0    ;CLEAR SCREEN

0210            233E    ;DO SUBROUTINE - GENERATE TARGETS
  12    LOOP1:  A3BC    ;POINT I TO BALL PATTERN
  14            DCD1    ;SHOW BALL @ VC,VD
  16    LOOP2:  A3BC    ;POINT I TO BALL PATTERN
  18            DCD1    ;SHOW BALL @ VC,VD
  1A    LOOP3:  8CA4    ;LET VC=VC+VA (MOVE BALL ALONG X)
  1C            8DB4    ;LET VD=VD+VB (MOVE BALL ALONG Y)
  1E            600F    ;LET V0=0F

0220            E0A1    ;SKIP NEXT IF NOT KEY F
  22            2300    ;DO SUBROUTINE - PLACE SLASH ON DISPLAY
  24            6001    ;LET V0=01
  26            E0A1    ;SKIP NEXT IF NOT KEY 1 ('GIVE-UP' KEY)
  28            1278    ;GO SHOW SCORE AND END ON COMMAND
  2A            60FE    ;LET V0=FE FOR UPCOMING XOR COMMANDS
  2C            4C00    ;SKIP IF VC <> 00 (LEFT)
  2E            8A03    ;LET VA=VA XOR V0 (GO RIGHT)

0230            4C3F    ;SKIP IF VC <> 3F (RIGHT)
  32            8A03    ;LET VA=VA XOR V0 (GO LEFT)
  34            4D00    ;SKIP IF VD <> 00 (TOP)
  36            8B03    ;LET VB=VB XOR V0 (GO DOWN)
  38            4D1F    ;SKIP IF VD <> 1F (BOTTOM)
  3A            8B03    ;LET VA=VA XOR V0 (GO UP)
  3C            A3BC    ;POINT I TO BALL PATTERN
  3E            DCD1    ;SHOW BALL @ VC,VD

0240            3F01    ;SHIP IF HIT SOMETHING
  42            1216    ;         ELSE GO LOOP2
  44            236A    ;CHECK IF IT WAS A TARGET
  46            3F01    ;SKIP IF IT WAS (VF=01 = TARGET HIT)
  48            1254    ;NOT TARGET - GO BOUNCE OFF SLASH
  4A            FF18    ;BEEP!
  4C            7401    ;LET V4=V4+1 (INCREASE SCORE)
  4E            9490    ;SKIP IF V4 <> V9 (# TARGETS SELECTED)

0250            1278    ;GO SCORE AND END -- YOU WIN!
```

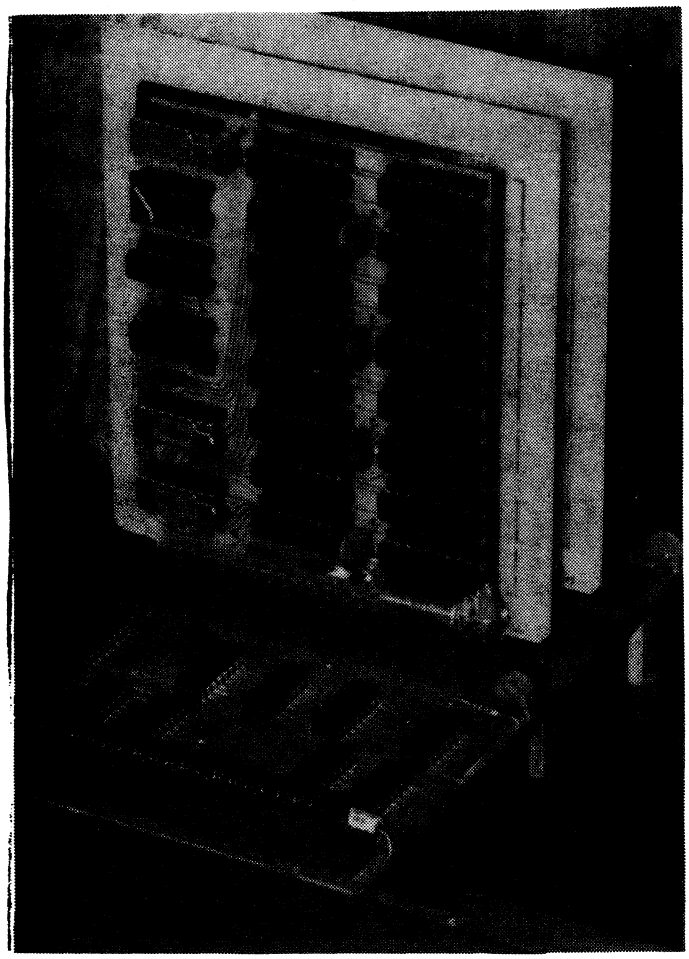
```
  52              121A    ;GO LOOP3 - STILL SOME TARGETS LEFT
  54              3BFF    ;SKIP IF VB=-1 (GOING UP NOW)
  56              125E    ;GO CHECK NEXT CONDITION
  58              6B00    ;LET VB=00 (GO RIGHT)
  5A              6A01    ;LET VA=01 ("    "  )
  5C              1216    ;GO TO LOOP2
  5E              3B01    ;SKIP IF VB=+1 (GOING DOWN NOW)

0260              1268    ;GO CHECK NEXT CONDITION
  62              6B00    ;LET VB=00 (GO LEFT)
  64              6AFF    ;LET VA=FF ("   "  )
  66              1216    ;GO TO LOOP2
  68              3A01    ;SKIP IF VA=+1 (GOING RIGHT NOW)
  6A              1272    ;GO CHECK NEXT CONDITION
  6C              6A00    ;LET VA=00 (GO UP)
  6E              6BFF    ;LET VB=FF ("  " )

0270              1216    ;GO TO LOOP2
  72              6A00    ;LET VA=00 (GO DOWN)
  74              6B01    ;LET VB=01 ("   "  )
  76              1216    ;GO TO LOOP2
  78    SCORE:    00E0    ;ERASE SCREEN
  7A              6001    ;LET V0=1 (VX)
  7C              6102    ;LET V1=2 (VY)
  7E              A3BC    ;POINT I TO TARGET PATTERN

0280              D011    ;SHOW TARGET (AS A SCORING LABEL)
  82              6014    ;LET V0=14
  84              6100    ;LET V1=00
  86              A3C6    ;POINT I TO LARGE SLASH PATTERN
  88              D015    ;SHOW '/' (AS SCORING LABEL)
  8A              6000    ;LET V0=00
  8C              6109    ;LET V1=09
  8E              A3BA    ;POINT I TO SMALL SLASH PATTERN

0290              D013    ;SHOW SLASH (AS SCORING LABEL)
  92              6C04    ;LET VC=04
  94              6D00    ;LET VD=00
  96              8040    ;LET V0=V4 (YOUR SCORE)
  98              2392    ;DO SUB - SHOW # TARGETS HIT
  9A              6C1B    ;LET VC=1B
  9C              8090    ;LET V0=V9 (# TARGETS SELECTED)
  9E              2392    ;DO SUB - SHOW # TARGETS SELECTED

02A0              6C04    ;LET VC=04
  A2              6D08    ;LET VD=08
  A4              8050    ;LET V0=V5 (# SLASHES USED)
  A6              2392    ;DO SUB - SHOW # SLASHED USED
  A8              12A8    ;STOP HERE -- END OF GAME
  AA    PATCH:    C03F    ;LET V0=RND # (00..3F)
  AC              6E03    ;LET VE=03
  AE              8E05    ;SUBTRACT VE-V0
```

```
02B0            3F00    ;SKIP IF V0>3
  B2            12AA    ;TOO LOW, TRY AGAIN
  B4            6E3C    ;LET VE=3C
  B6            8E05    ;SUBTRACT VE-V0
  B8            3F01    ;SKIP IF V0<=3C
  BA            12AA    ;TOO HIGH, TRY AGAIN
  BC            C11F    ;LET V1=RND # (00..1F)
  BE            6E03    ;LET VE=03

02C0            8E15    ;SUBTRACT VE-V1
  C2            3F00    ;SKIP IF V1>3
  C4            12BC    ;TOO LOW, TRY AGAIN
  C6            6E1C    ;LET VE=1C
  C8            8E15    ;SUBTRACT VE-V1
  CA            3F01    ;SKIP OF V1<=1C
  CC            12BC    ;TOO HIGH, TRY AGAIN
  CE            4111    ;SKIP IF V1<>11 (KEEP STARTING LANE OPEN)

02D0            12AA    ;GO TRY ANOTHER X,Y PAIR
  D2            8E06    ;SHIFT V0 RIGHT TO TEST LSB (EVEN/ODD TEST)
  D4            3F01    ;SKIP IF V0 IS ODD
  D6            134E    ;RETURN FROM PATCH
  D8            8E16    ;ELSE TEST V1 FOR EVEN/ODD
  DA            3F01    ;SKIP IF V1 IS ODD
  DC            12AA    ;NUTS, TRY AGAIN (IF VX ODD, VY MUST BE EVEN)
  DE            134E    ;RETURN FROM PATCH

02E0 - 02FF             ;UNUSED AREA

0300            13CC    ;GO TO PATCH @ 03CC
  02            80C5    ;LET V0=V0-VC
  04            3F00    ;SKIP IF VC>=2 (IE 1<VC)
  06            00EE    ;RETURN, VX TOO LOW
  08            603D    ;LET V0=3D
  0A            80C5    ;LET V0=V0-VC
  0C            3F01    ;SKIP IF VC<=3D
  0E            00EE    ;RETURN, VX TOO HIGH

0310            6002    ;LET V0=02
  12            80D5    ;LET V0=V0-VD
  14            3F00    ;SKIP IF VD>=3
  16            00EE    ;RETURN, VY TOO HIGH
  18            601D    ;LET V0=1D
  1A            80D5    ;LET V0=V0-VD
  1C            3F01    ;SKIP IF VD<=1D
  1E            00EE    ;RETURN, VY TOO HIGH

0320            600F    ;LET V0=0F FOR KEY CHECK
  22            E0A1    ;SKIP IF KEY PRESSED <> 0F
  24            1322    ;LOOP TILL KEY IS RELEASED
  26            80C0    ;LET V0=VC(VX)
```

```
  28            81D0    ;LET V1=VD(VY)
  2A            70FF    ;LET V0=V0-1
  2C            71FF    ;LET V1=V1-1
  2E            A3BA    ;POINT I TO SLASH PATTERN

0330            D013    ;DISPLAY SLASH @ V0,V1
  32            3F01    ;SKIP IF HIT SOMETHING
  34            133A    ;GO CONTINUE
  36            D013    ;ERASE THE CLASHING SLASH
  38            00EE    ;EXIT - NO SLASH THIS TIME
  3A            7501    ;LET V5=V5+1 (# SLASHES)
  3C            00EE    ;RETURN
  3E            6200    ;INDEX=V2=00

0340            6300    ;LOOP COUNTER=V3=00
  42            12AA    ;GO PATCH
  44            0000    ;PATCHED OUT SECTION
  46            0000
  48            0000
  4A            0000
  4C            0000
  4E            A400    ;POINT I TO TOP OF ARRAY

0350            F21E    ;ADD I+V2 TO FORM ADDRESS
  52            F155    ;STORE V0,V1 @ M(I),M(I)+1
  54            A3BC    ;POINT I TO TARGET PATTERN
  56            D011    ;SHOW TARGET
  58            3F01    ;SKIP IF HIT (VF=01)
  5A            1360
  5C            D011    ;REPAIR DAMAGE BY HIT
  5E            1342    ;TRY AGAIN.  DUPLICATES NOT LEGAL

0360            7202    ;LET V2=V2+2 (INDEX TO ARRAY)
  62            7301    ;LET V3=V3+1 (LOOP COUNTER)
  64            5390    ;SKIP NEXT WHEN V3=V9= # TARGETS
  66            1342    ;REPEAT UNTIL ALL TARGETS SHOWN
  68            00EE    ;RETURN
  6A            6200    ;LET V2=00- LOOP COUNT
  6C            A400    ;POINT I TO FIRST X,Y COORDINATE
  6E            F165    ;LET V0=VX,V1=VY OF A TARGET

0370            50C0    ;SKIP IF V0=VC (BALL VX)
  72            1378    ;NOPE, NOT THIS ONE
  74            91D0    ;SKIP IF V1 <> VD (BALL VY)
  76            1384    ;YEP! GOT ONE; GOTO EXIT
  78            7201    ;LET V2=V2+1 (TRY ANOTHER)
  7A            5290    ;BUT SKIP IF V2=V9 (# TARGETS)
  7C            136E    ;GO TRY ANOTHER X,Y PAIR
  7E            6F00    ;SET FLAG=00 (AWWW, NOT A TARGET)

0380            00EE    ;RETURN
  82            0000    ;(UNUSED)
```

```
  84          A400    ;POINT TO TOP OF ARRAY AGAIN
  86          F21E    ;LET I = I+V2*2
  88          F21E    ;(             )
  8A          60FF    ;LET V0=FF
  8C          F055    ;ERASE THIS TARGET FROM ARRAY
  8E          6F01    ;SET FLAG =01 (YAAY! GOT ONE)

0390          00EE    ;RETURN
  92          A400    ;POINT TO WORK SPACE @ 0400
  94          F033    ;LET MI=3 DECIMAL DIGIT OF V0
  96          F265    ;LET V0:V2= THOSE DIGITS
  98          F029    ;LET I = PATTERN FOR DIGIT IN V0
  9A          DCD5    ;SHOW IT @ VC,VD
  9C          7C05    ;LET VC=VC+5
  9E          F129

03A0          DCD5    ;SHOW SECOND DIGIT
  A2          7C05
  A4          F229
  A6          DCD5    ;AND THE THIRD
  A8          00EE    ;RETURN
  AA          0000    ;V0,V1 INITIAL VARIABLE VALUES
  AC          0000    ;V2,V3
  AE          0000    ;V4,V5

03B0          0000    ;V6,V7
  B2          0000    ;V8,V9
  B4          0100    ;VA,VB
  B6          2011    ;VC,VD
  B8          0000    ;VE,VF
  BA          2040    ;SLASH
  BC          8000    ;  AND BALL/TARGET PATTERNS
  BE          F888    ;QUESTION MARK

03C0          0830    ;   "    "
  C2          2020    ;   "    "
  C4          0020    ;   "    "
  C6          0810    ;LARGE SLASH (USED IN SCORE)
  C8          2040    ;   "    "
  CA          8000    ;   "    "
  CC          4A00    ;IF GOING HORIZONTALLY THEN SKIP
  CE          13D8    ;GO CHECK VERTICAL

03D0          80C6    ;SHIFT VC RIGHT TO TEST LSB
  D2          3F00    ;SKIP IF VC IS EVEN (LSB=0)
  D4          00EE    ;RETURN - NO SLASH
  D6          13E0    ;GO TO EXIT PATCH
  D8          80D6    ;TEST LSB VD BY SHIFTING RIGHT
  DA          3F01    ;SKIP IF LSB =1 (VD IS ODD)
  DC          00EE    ;RETURN IF VD IS ODD
  DE          6001    ;PATCHED INSTRUCTION
03E0          1302    ;GO BACK - SO FAR SO GOOD
```

8K RAM BOARD FOR YOUR RCA VIP



- *USES 2114 RAMS
- *EACH 4K BLOCK SEPARATELY ADDRESSABLE
- *G10 GLASS BOARD
- *DOUBLE SIDED 2 OZ COPPER
- *SOLDER PLATED
- *GOLD PLATED CONTACTS
- *4-1/2" W x 5-3/8" H
- *PLUGS INTO VIP EXPANSION CONNECTOR OR SYSTEM EXPANSION BOARD
- *DRAWS APPROXIMATELY 600 mA FROM +5 V LINE
- *BARE BOARD WITH DATA $ 49.00
- *ASSEMBLED AND TESTED $ 149.00

CALIFORNIA RESIDENTS ADD STATE SALES TAX

WITH THE RCA VP-575 SYSTEM EXPANSION BOARD YOU CAN PLUG IN TWO OF THESE 8K RAM BOARDS, ONE 12K ROM BOARD* PROGRAMMED WITH YOUR VERSION OF RCA VP-701 FLOATING POINT BASIC, AND STILL HAVE ROOM FOR A VP-590 COLOR BOARD AND A SPARE.

*FUTURE PRODUCT

SEND CHECK OR MONEY ORDER TO:

G. J. KRIZEK
722 N. MORADA AVENUE
WEST COVINA, CA 91790

MACHINE CODE

P. V. Piescik, Prop., Cuddly Software, 157 Charter Road, Wethersfield, CT 0618

While there may be doubt in the minds of some members, I feel that if a newcom to the computing hobby can't find tutorials in a club newsletter, what good is joining? Finding programs to run is fine, but giving a man a fish feeds him once--teach him to fish and you'll allow him to feed himself for life! Let's not deny VIPHCA novices the joy of accomplishment, the creative outlet, the sweet victory of conquering the "8-bit Cyber-Beast," found when it does our bidding!

Programming is a jigsaw puzzle activity--we take a problem, part of the real world, and cut it up into pieces. The instructions and data formats available to us on a given machine (or in a given language) are the shapes we can legally use. While these features vary among languages and machines, the two biggest differences between machine language and higher level languages (for our purposes, assembly language is on a higher level) are that 1) the "pieces" are smaller, and 2) "housekeeping" is our responsibility. The smaller pieces mean that we have better control over the process (higher resolution, as it were); the housekeeping is somewhat of a pain at times, and probably, the source of trouble for novices.

Assumptions: You have an 1802 machine with at least 1 page of RAM (but we'll consider in our programs that there are at least 2 pages), you have a copy of MPM-201 and have read it, regardless of how much you understand. We'll also assume that you have a primitive monitor which allows you to read (examine) memory, write memory, and save and load cassettes.

Problem 1: Add two numbers. It seems insignificant, which is good--no one will be overwhelmed by the problem and we can concentrate on the machine stuff.

Since the manual contains wholesale explanations of the architecture and instructions, I won't repeat that here. Rather, as we solve problems, I'll cover what we need to know immediately, a little piece at a time, to avoid overwhelming complexity.

There are a few very basic characteristics of the 1802 to consider right from the start. Writing to memory replaces the old contents of a memory location, so we don't want to write to a location if we're going to need the old contents later. Reading memory merely copies the contents of a location, usually into D. We can read the same contents from the same location later, provided we haven't written to it between readings. All of the registers also work this way, although different instructions are used to read and write registers. All memory addressing is indirect, i.e., one of the scratchpad registers contains the address we want (we have to put it there) and we specify the location of a read or write by indicating which register the 1802 will use to address memory. The last thing to remember about memory is that until we have stored what we want in any location, by keying from the hexpad, loading tape, or the action of our program, the contents of the location are unknown leftovers (garbage) from some earlier event. The only way we can trust that memory contains what we expect is to put it there explicitly (initialize it). Initialization also applies to registers, and ALL we know about the state of registers and memory when our program starts (which is where we begin planning) is that I, X, P, Q, and R0 contain 0 (the value, regardless of how many bits are used), and IE contains 1. Everything else is an unknown leftover.

What the 1802 will do, for us or to us, is fetch instructions from memory and execute them, one after the other, forever. We can alter the sequence of instructions by branching or calling subroutines instead of proceeding to the next memory location; but we'll worry about that later. For now, the operative word is "forever," which means that just because the 1802 reaches the logical end of our program, it isn't going to stop cycling! If we don't do something about stopping it's progress, it will continue to fetch and execute "instructions." Only the data fetched from memory probably won't be a logical sequence of instruction (which is the definition of a program), but the 1802 can't tell by looking! As it proceeds through this unplanned sequence of instructions, we have lost control over the machine which is now a runaway, and we've bombed. I mention how to stop the program when we get to it in our solution to Problem 1.

The memory address of the first and all subsequent instructions must be in a register, since that is the only way memory can be addressed. One of the scracthpads is designated as the Program Counter (PC) by the 4-bit P register; the PC may also be called R(P), and while we usually know the value of P, there may be times when we don't. Since we know at the start that P=0, R0 is the PC, and we also know that it contains 0, so our first instruction must be at 0000.

Let's start by adding two constants, 1 and 2, which can be anywhere in memory in locations we'll call ONE and TWO, and we'll store the result in a third location called SUM. The ADD (F4) instruction has two operands, one in the accumulator and one in memory at M(R(X)). (X and R(X) work just like P and R(P) to designate which register is used to Index memory.) We'll also use 3 registers to point to ONE, TWO and SUM (by containing those addresses). We'll also note that almost all data and addresses which we want to move must go through D, so we must do all of our register set-ups (housekeeping) before we move our addends or sum to D, since we'd otherwise lose the data by doing the remaining set-up(s). With RC, RD, and RE pointing to ONE, TWO, and SUM respectively:

```
ADDR  INSTR  ASSEMBLY STATEMENT

0000  F8 --  LDI A.1(ONE)
0002  BC     PHI C         ..RC.1 <-- HIGH ORDER ADDR OF ONE
0003  F8 --  LDI A.0(ONE)
0005  AC     PLO C         ..RC.0 <-- LO ADDR OF ONE
0006  F8 --  LDI A.1(TWO)
0008  BD     PHI D         ..RD.1 <-- HI ADDR OF TWO
0009  F8 --  LDI A.0(TWO)
000B  AD     PLO D         ..RD.0 <-- LO ADDR OF TWO
000C  F8 --  LDI A.1(SUM)
000E  BE     PHI E         ..RE.1 <-- HI ADDR OF SUM
000F  F8 --  LDI A.0(SUM)
0011  AE     PLO E         ..RE.0 <-- LO ADDR OF SUM
0012  EE     SEX E         ..X=E, RX IS RE; END HOUSEKEEPING
```

OK, the first column is the location of the first byte of each instruction or data item in memory; the second column is the machine language instruction; the third column is what the assembly language statement would be, were we using an assembler. Note that since the 1802 is strictly an 8-bit machine, we have to load the 16-bit registers by halves, using two instructions per half. The first instruction (F8 --) moves a value to D; the second instruction (Bn or An) moves the value to the register-half.

We haven't yet decided where ONE, TWO and SUM will be, so we can't put their addresses in the F8-- instructions, but the "--" will remind us to save the room now and to fill it in later. Let's put these locations right after the end of the program; this decision still doesn't allow us to fill in the blanks, since we aren't done and don't know how long the program will be. At this point, even an assembler wouldn't know that; it, too, would come around again to plug in values. The difference is that we are doing the work instead of an assembler program.

The last of the housekeeping details was to set X=E, so the operand which is at M(R(X)) will be at M(RE), which is SUM. We're going to clobber the contents of SUM anyway, so we might as well use it temporarily instead of a fourth memory location we would call TEMP. What we cannot do here, is leave X=0, since R0 is the PC. That would mean storing the temporary value within our program, altering the code, which is a practise we'd like to avoid so the original program will be intact for the next run. Then, too, storing data with a 50 instruction when P=0 will drop the data into memory in the location from which the next instruction will be fetched. Not only does this clobber an existing instruction, but that DATA will be EXECUTED as the next instruction, very likely bombing us!

Let's continue with the coding:

```
0013  0C     LDN C          ..GET DATA AT ONE
0014  5E     STR E          ..STORE AT M(RX)
0015  0D     LDN D          ..GET DATA AT TWO, INTO D
0016  F4     ADD            ..D NOW CONTAINS RESULT
0017  5E     STR E          ..STORE RESULT AT SUM
```

We have now accomplished our logical objective of adding two numbers from memory and storing the result in a third location. This is where we have to stop the 1802 from proceeding further:

```
0018  30 18  BR *           ..BRANCH TO THIS LOCATION
```

The "*" is assembly shorthand for the location of the current instruction (always the first byte of two- or three-byte instructions). This instruction will branch to itself forever, effectively halting the 1802's progress, although we should note that this is a "busy stop." We could also use an IDL (00) instruction, ONLY if we know that no interrupts can occur or interrupts are masked off (IE=0), and that no DMA's will be requested (DMA's are unmaskable, i.e., always enabled). Or, IF we know which register is the PC, we could use a 2P instruction (in this case, 20) which is a 1-byte equivalent to "BR *". A very common misconception is that if the 1802 isn't going to stop here without the BR *, we'll catch it manually with the Reset button/switch. HA! This program takes only 36 machine cycles, so it will be over in less than 164 microseconds, so you'd better be wearing your red longjohns from the spaceship!

Here's the end of the program, so we'll place the data where we decided:

```
001A  01     ONE: DC 1      ..INITALIZED CONSTANT
001B  02     TWO: DC 2      ..INITIALIZED CONSTANT
001C  xx     SUM: ORG *     ..NOT INITIALIZED
```

We have called these locations ONE, TWO and SUM, but the names are only useful as reminders of how we are using the locations. An assembler would evaluate these symbols as follows: ONE = 001A, TWO = 001B, SUM = 001C; it would then plug in the high- or low-order byte of the appropriate value where it's needed in the machine code. We're not using an assembler, however, and all addressing on the 1802 is absolute, i.e., the machine code must contain the actual values of the addresses. (The alternative is "relative" addressing, a capability not found on the 1802, where addresses may be specified by a displacement which is added to, or subtracted from, a base address at the start of the program, or the PC. These are called Base-relative and PC-relative addressing.) We must also be aware that ONE, TWO and SUM refer to memory locations, not to the values contained there. So, ONE = 001A, but (ONE) = 01, etc. "(ONE)" is shorthand for "the contents of ONE." Let's fill in those blanks we left earlier:

```
0000  F8 00
0002  BC
0003  F8 1A
0005  AC      ..(RC) = 001A
0006  F8 00
0008  BD
0009  F8 1B
000B  AD      ..(RD) = 001B
000C  F8 00
000E  BE
000F  F8 1C
0011  AE      ..(RE) = 001C
0012  EE
0013  0C
0014  5E
0015  0D
0016  F4
0017  5E      ..(SUM) = (ONE) + (TWO)
0018  30 18   ..STOP
001A  01      ..ONE
001B  02      ..TWO
001C  xx      ..SUM
```

We can now enter this program at 0000-001B (we don't initialize 001C, so why bother?) and run it. When we then examine memory, we'll find that the contents of 001C is 03. If not, verify that the program was entered correctly; if it was and the result is wrong, you have hardware problems.

The program, including data, is 29 (decimal) bytes long, and only 5 bytes of that are used to perform our task. Housekeeping (setting up registers and stopping the program) accounts for 21 bytes; it looks like a lot, but this is a very small problem and housekeeping will look smaller when our problems are bigger. However, one of the nice things about machine code is that we have many opportunities to "cheat" without really using dirty tricks. If we notice that the high-order bytes of RC, RD, and RE all contain '00' because all the data is on the same page, we can eliminate 4 bytes and save 4 machine cycles at execution time by rearranging a few instructions:

```
0000  F8 00
0002  BC BD BE     ..ALL HI-ADDRS
0005  F8 16 AC     ..A.0(ONE)
0008  F8 17 AD     ..A.0(TWO)
000B  F8 18 AE     ..A.0(SUM)
000E  EE           ..THIS USED TO BE AT 0012
000F  0C
0010  5E
0011  0D
0012  F4
0013  5E
0014  30 14        ..STOP
0016  01           ..ONE
0017  02           ..TWO
0018  xx           ..SUM
```

We can save another 2 bytes by realizing that if X=C or X=D, one of the operands will already be at M(RX) and we won't have to move it. Let's set X=C this time:

```
0000  F8 00
0002  BC BD BE
0005  F8 14 AC
0008  F8 15 AD
000B  F8 16 AE
000E  EC           ..X=C, DELETE 0C 5E INSTRS
000F  0D
0010  F4
0011  5E
0012  30 12        ..THIS WAS AT 0018 THE 1ST TIME
0014  01           ..ONE
0015  02           ..TWO
0016  xx           ..SUM
```

Try this to verify that it works. It does, because the operand at M(RX) is only read from memory by the ADD (F4) instruction (which does not alter memory) and the result of the ADD is in the accumulator. The operand at M(RX) is read directly from memory into the ALU (Arithmetic Logic Unit).

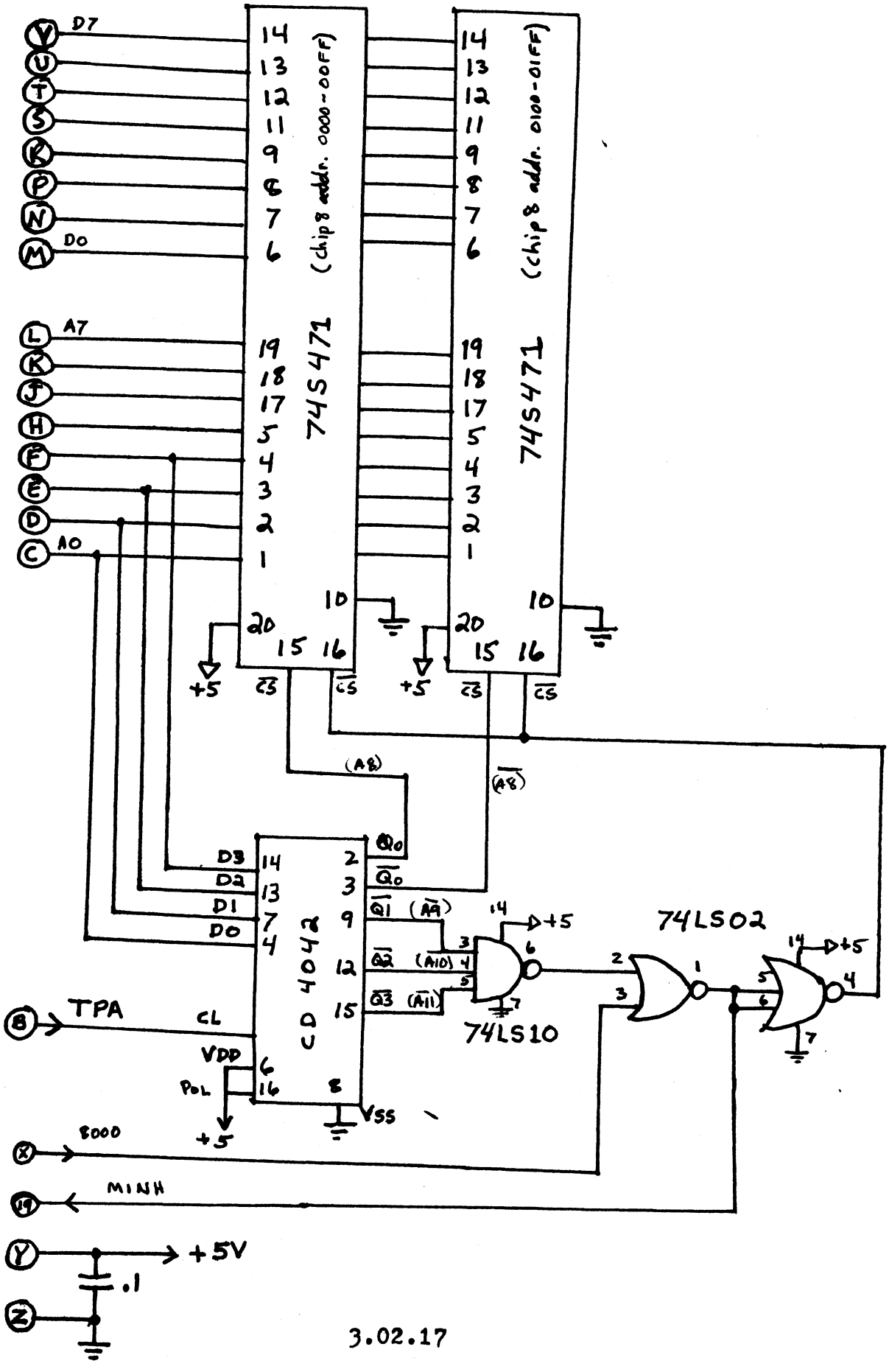
If we're really sharp, hate housekeeping, and realize that our data occupies successive locations in memory, we can eliminate the need for all but one of our registers pointing to data! Let's use RE, which will also be RX:

```
0000  F8 00  LDI A.1(ONE)
0002  BE     PHI RE       ..R-FORM OF REGISTER NOTATION
0003  F8 0D  LDI A.0(ONE)
0005  AE     PLO RE       ..(RE) = A(ONE)
0006  EE     SEX RE       ..X = E
0007  4E     LDA RE       ..GET (ONE), POINT TO TWO
0008  F4     ADD          ..(D) = (ONE) + (TWO)
0009  1E     INC RE       ..POINT TO SUM; COULD ALSO USE IRX '60'
000A  5E     STR RE       ..(SUM) = (D) = (ONE) + (TWO)
000B  30 0B  BR *         ..BUDY STOP
000D  01     ONE: DC 1
000E  02     TWO: DC 2
000F  xx     SUM: ORG *
```

We have chopped this program from it's original 29-byte memory requirement to a mere 16 bytes, and from 36 cycles to 18 (not counting the busy stop instruction). If we measure program efficiency in terms of memory bytes times machine cycles, we are 3.63 times as efficient as when we started. What we traded, however, was the unrestricted location of any datum in memory in the first program; to the constraint of having all data on the same memory page but not necessarily adjacent, in the second program; in the last program, the data must be in contiguous locations (although we can cross a page boundary), with the operands (addends) preceding the result (sum). At the same time, we have gained some convenience if we need to modify the program. If we do not load this program on page 00, in the first version we have 3 places to change so the pointers will be on the new page. In any of the other versions we have only 1 place to change. We aren't saving that much work, but with only 1 byte to change, there is no possibility of missing some of the changes, which would prevent the program from operating on the correct data.

OK. I think we've beaten this one to death. I intend to continue with MACHINE CODE in each issue until we reach a point where your efforts can be directed at solving problems instead of dis-solving the 1802. Since I was prompted to write this after seeing a letter in VIPER 3.01, and I have no other way of knowing whether I'm going over your head or under your shoes, give me some feedback! If you classify yourself as a beginner, drop me a line telling if I'm going too fast, too slow, just right; and if you have any particular topics/questions you'd like to see covered. This series is for "lost sheep," so I'll tune it to your needs based on your feedback; my "reply" will be seen in the content of this series. If you know what you're doing and don't need this, don't write (unless you find gross errors, but even these little programs have been tested)!

CHIP-8 PROM Board Circuit



3.02.17

CHIP-8 PROM Board for the VIP

By Bob Casey

The accompanying drawing shows how to construct a plug-in board with CHIP-8 in PROM. The PROMs used are 256 by 8 74S471s. It's pretty much straightforward, but some explanation seems appropriate. The 8000 input makes sure the PROMs are "tri-stated" into high impedance when the operating system ROM is addressed by the 1802. The MINH output shuts off the RAM at addresses 0000 to 01FF.

Having a CHIP-8 PROM card is convenient because you don't have to load CHIP-8 from tape; just plug in the card. But don't insert or remove it with power on. A bug in a program can't destroy CHIP-8 when it's on PROM! (But then you no longer have the ability to quickly change code in the interpreter itself.-RS)

BOOMERANG PUZZLE
(Requires Tiny BASIC Board)

```
10   CLS
15   PRINT
16   PRINT
17   PRINT
20   PRINT "BOOMERANG PUZZLE"
30   PRINT
40   PRINT "PLEASE THINK"
50   PRINT "OF A NUMBER"
60   PRINT "FROM 1 TO 100"
63   LET I=Ø
64   LET I=I+1
65   IF I<4 THEN 64
70   PRINT
80   PRINT "YOUR NUMBER"
90   PRINT "DIVIDED BY 3"
100  PRINT "HAS A REMAINDER"
105  INPUT A
108  PRINT
110  PRINT "DIVIDED BY 5"
120  PRINT "HAS A REMAINDER"
125  INPUT B
128  PRINT
130  PRINT "DIVIDED BY 7"
135  PRINT "HAS A REMAINDER"
138  INPUT C
140  PRINT
145  PRINT "NOW LET ME SEE..."
200  LET D = 70*A+21*B+15*C
210  IF D<= 105, THEN 240
220  LET D=D-105
230  GOTO 210
240  PRINT
250  PRINT "YOUR NUMBER IS"
255  PRINT D
256  PRINT "RIGHT?"
260  GOSUB 500
280  IF A=0 THEN 380
300  IF A=2 THEN 320
310  PRINT "I THOUGHT SO!"
315  GOTO 330
318  PRINT
320  PRINT "YOU MUST BE"
321  PRINT "MISTAKEN"
330  PRINT
340  PRINT "DO YOU WANT TO"
350  PRINT "TRY AGAIN?"
360  GOSUB 500
370  IF A=1 THEN 30
380  PRINT "OK - GOODBY!"
390  END
500  REM INPUT SUB
510  PRINT "PLEASE ENTER"
520  PRINT "KEY Ø FOR EXIT"
530  PRINT "KEY 1 FOR YES"
540  PRINT "KEY 2 FOR NO"
550  INPUT A
560  IF A>2 THEN 590
570  IF A<0 THEN 590
580  RETURN
590  CLS
595  GO TO 500
```

VIP Hobby Computer Association
32 Ainsworth Avenue
East Brunswick, NJ 08816

A Final Word:

As of a month or two ago, the folks at ARESCO (Box 1142, Columbia, MD 21044) had some VIP hardware left over and were willing to sell the items at close-out prices. The following items may or may not still be available:

VP-620 $15, VP-585 $11, VP-590 $52, VP-595 $23, VP-550 $37, VP-700 $29, VP-710 $6, VP-320 $5, MPM 201B $5. These prices do not include shipping, so add 10% or so, excess will be refunded. Use only street addresses for UPS.

ARTICLES STILL PENDING FOR COMING ISSUES:

1. Simple Music Program Part 2 by Udo Pernisz
2. COSMAC VIP Autocall by George Gadbois
3. VIP Operating System for ELF by Leo Hood

Last minute item:
Tom Swan's book, Programmers Guide to the 1802 is now out. 156 pgs. from Hayden Book Co. VIPER will review it soon.