VIPER

December 1982 - January 1983

Volume 4, Number 5 Journal of the VIP Hobby Computer Assn.
The VIPER was founded by ARESCO, Inc. in June 1978

****

Contents

EDITORIAL

There were no names placed in nomination for the Directorship of VIPHCA, per the election announcement in the last VIPER. Since I indicated that I'd be willing to serve another term, I guess that I've been nominated, and failing to have any opposition, elected. There was mail from some of you indicating that you were satisfied with yours truly as Director, so I guess that is a kind of acclamation! Seriously, if any of you are unhappy about VIPHCA, please let me know. I can't promise the world, but I'll listen and do what I can.

Some of you may have noticed that VIPER 4.04 had extra postage on it. It seems that when the issues were placed on the postal scale they weighed in at over 2 ounces each. And over 2 ounces means that an extra 17 cents worth had to be applied. I don't know why, since that VIPER had the same number of pages as usual, came from the regular printer, used the same paper, and so on, as any of the other VIPER issues of this year. Maybe the ink was heavier. So to prevent that problem from besetting this issue, I've reduced the number of pages.

Also missing from this VIPER is Paul Piescik's Machine Code Column. Sadly, I got a letter from Paul saying that his new employer here in New Jersey considered that his writing for the VIPER would be a conflict of interest, and as a result, he would not place his job in jeopardy by continuing the column. I do understand, but I am sure that we will all miss his contribution to the VIPER. I know how much work Paul put into his material, and I really very much appreciate what he has do for us. If any of you out there have ever considered doing a machine code column, now is the time to volunteer! Paul has left us with a parting gift, however. As most of you know, Paul was in business for a while as Cuddly Software. That business was discontinued so time ago, but Paul still got occasional requests for his programs. Accordingly, Paul has released into the Public Domain all of his material with the "CS" designation: CSOS, CSTP, CSPIOP, CSIO, CSAP. This means that you may legally make copies of these programs for others. The "STARSHIP" program is NOT--repeat--NOT included in this release. Paul sold that program under license. If you are interested in "STARSHIP" contact: John R. Powers III, 1609 Serpa Dr., Milpitas, CA 95035.

Ray

Raymond C. Sills,
Director, VIPHCA

Special Routines for Morse Code
from Standard CHIP-8
Steven Vincent Gunhouse

I have met several people who had a desire to write programs in CHIP-8 which would send the International Morse Code. While this is not complicated, with the timing requirements of morse code it becomes practically impossible to have the program do anything else while sending code. With my knowledge of 1802 machine language, I decided it should be possible to write a routine to take care of this with a minimum of outside effort and programming. After I had written my previous morse code program (see VIPER 4.01), I realized that this would not be too difficult, but it would still tie up the machine for too long. There had to be another way.

Finally I realized what had to be done. The present CHIP-8 tone commands use the interrupt routines for timing. It should be possible to put all the necessary timing routines in the interrupt routine and thus free the program from time commitments as much as possible. I went ahead to try and write such a routine.

Immediately I had a problem. Consulting my previous programs I found I would need three 8-bit registers in order to carry out these functions. That was fine until I looked at the register allocations for CHIP-8. I did not seem to have any I could safely use without eliminating too many for most machine language subroutines. This might take a little more work than I thought. I decided to take a closer look at the routines in CHIP-8 to see if there were any registers I could conveniently eliminate.

I did not have much luck at first. It looked like I would need to do a complete revision of CHIP-8 to free up any registers at all, but then I looked at the interrupt routine itself, which I had to rewrite anyway. Suddenly I had my three registers, in places no experienced programmer would ever dream of using in his machine language subroutines. In fact, normally he couldn't. I would use R(0) and R(B.0).

Even with that decided, it still was not going to be easy. First I had to decide to give the Morse Code output priority for the Q line. Then there were innumerable technical difficulties to be worked out. Finally, a few of the standard instructions had to be moved to make room for the output commands, but I could not move any location-dependent commands like FXHH. When I was finished with my new program about a month later, it was one page longer than CHIP-8 and responded to all the regular instructions. I decided that CHIP-8M would be an appropriate descriptive name, and then proceeded to write some very simple programs for it.

It is remarkably easy to write morse code programs in CHIP-8M. Because it includes an ASCII keyboard read command, a simple morse code keyboard program with a one-byte buffer takes exactly three commands (6 bytes of program memory). A slightly more complicated program will still be relatively simple, especially compared to writing a morse code program in normal CHIP-8. Due to the lengthened interrupt routine, normal programs may be a little slower, but for morse code it is hard to beat it.

Some changes were necessary on every page of CHIP-8, though most of them must of course fall on page 02. Programs start at location 0300, which allows a great deal of space for most of your programs even in a 2K machine. Incidently, I could not find any easy way to have the speed under machine control, though there is always the "brute force" method. This will be detailed later. The changes necessary to CHIP-8 are as follows:

```
@000A 02             @000D 08            Address of start of interrupt routine
@0058 02             @0068 5D            New address of 8XYN instruction
@0100 E6 63 26 D4                        Output port control instruction FX00
@01BC  91 BC 06 F9                       Morse Code output instructions FXBC and
@01C0  80 AC FC 20 3B CF 30 DO  91 BC 06 FA OF F9 90 AC      FXC8
@01D0  8B 3A DO 4C AB D4
@01F2  E6 3F F3 37 F5 6B  D4 00 02 7A

@0200  13 00 02 BO 20 12 42 70  C4 22 78 22 73 90 73 80
@0210  52 9B BO 94 AO 80 E2 20  AO E2 20 AO E2 20 AO E2
@0220  3C 15 80 E2 20 AO 19 20  AO 20 AO 98 32 32 AO 20
@0230  80 B8 42 AO 3A 02 02 BO  3A 4C 8B 32 56 76 52 FA
@0240  7F AB 32 45 7B 91 3B 49  FE BO 02 FE 20 F8 tc AO
@0250  90 3A 05 7A 30 05 88 32  53 28 7B 30 05 45 FA OF
@0260  3A 65 07 56 D4 AF 22 F8  D3 73 8F F9 FO 52 E6 07
@0270  D2 56 F8 FF A6 94 7E 56  D4 00 94 BO AO AB D4 70
@0280  8B 3A 80 F8 20 AO D4 00  8B 3A 88 90 3A 8B D4 00
@0290  3F 3E 3C 38 30 20 21 23  27 2F 06 11 15 09 02 14
@02A0  22 00 00 68 2A 2D 2A AE  80 00 00 00 00 31 00 00
@02B0  00 00 00 00 00 00 00 68  80 00 00 00 00 00 00 00
@02C0  01 35 52 00 00 00 00 5E  6D 6D 2A 00 73 28 6A 29
@02D0  3F 3E 3C 38 30 20 21 23  27 2F 47 55 00 00 00 4C
@02E0  22 06 11 15 09 02 10 OB  10 04 1E OD 12 07 05 OF
@02F0  16 1B OA 08 03 OC 18 OE  19 1D 13 6D 29 6D 30 40
```

The underlined location is the time constant (@024E). This value is used to determine the speed. Since it is officially fixed, you will need to set it by hand with the operating system, or else set I to this location and do an F055 with the desired speed value in VO. A time constant of 00 should be about 55wpm, though I have not verified this yet. Theoretically, OA is 5wpm, 04 is 11wpm, 03 is 14, 02 is just over 18, and 01 is 27½wpm. I am sorry that they are not at more convenient values, but there was no way to change them to any other values.

This set of routines incorporates seven new (or almost new) instructions in the CHIP-8 instruction set without changing any of the other instructions normally included. These instructions are as follows:

FX00	Output VX through output port
FXBC	Output Morse code of ASCII character in VX
FXC8	Output Morse code of Hex LSD of VX
FXF2	Input from standard (parallel) ASCII keyboard
027A	Initialize Morse output registers
0280	Send long space
0288	Wait for end of present character

These last two instructions were intended for use between the tone (FX18) and morse code (FXBC & FXC8) when these are used in the same program. Either an ASCII space (code 20 ASCII) or a "long space" must be sent before any valid code if the tone was being used, since the program will not automatically seperate these. Also, since tone is unusable during Morse output, it is necessary to senc an invalid character (such as 01 ASCII) or use the 0288 instruction before one tries to output a valid tone or read the Hex keypad.

I wanted to include one other instruction in these routines, but did not have enough room for it. If one is attempting to write a Morse keyboard with a longer-than-one byte buffer, this instruction will be required. The machine code for it would be:

```
@0XXX 8B 3A (XX+5) 15 15 D4          Skip Character completed
```

If you have no need for the FX00 instruction, you could used a condensed form (only valid anywhere on page 01) which would read as follows:

```
@01XX 8B 32 88 D4                      "       "         "
```

It would also be a good idea to have a routine to test the keyboard without waiting and possibly read it at the same time. These routines would be constructed as follows:

```
@0XXX 3F (XX+4) 15 15 D4             Skip on Keypress
@01XX E6 6B 37 88 D4                 Skip Keypress, get key (valid page 01 only)
```

Of course, this second 01XX routine could easily be placed at 01F2 in place of the standard key read routine. Once these changes are made, if desired, this interpretter becomes even more versatile.

The simplest demonstration program for this set of routines is probably the one-byte buffer Mosre Code keyboard. This consist of only the following three instructions:

```
@0300  F0F2 F0BC 1300
```

It can only accept characters as fast as it sends them, but will usually stay one character behind unless you are typing relatively slow.

If the previously listed modifications are made, a Morse code keyboard with automatic repeat and a 256 character buffer can be written without much difficulty. It will use only V0, V1, and V2, plus having the buffer at 0400. The code, in modified CHIP-8M, is as follows:

```
@0300 6100 6200 F0F2 1318  A400 F11E F055 7101
@0310 601E F015 9120 71FF  0100 132A 9210 132A
@0320 A400 F21E F065 F0BC  7201 F007 4000 1302
@0330 1318
```

It is admitted that this program is still very simple. It does not have easy speed control, backspacing in the buffer, or the ability to store some message for later use. It also does not display the characters at all. But it is only intended as a simple demonstration of the modified version of CHIP-8M. All other programs that you would like to construct are left up to you.

A note on my previous Morse code program which appeared in VIPER 4.01. It was brought to my attention that not all ASCII keyboards accept CONTROL and some number. The functions often reached through CONTROL 1, 2, &3 are called DC1, DC2, &DC3; these are also available on most keyboards as CONTROL Q, R, &S. If you have trouble finding these keys, consult the manual for your keyboard.

I expect to write again at a later date, possibly just before I return to good old BGSU. Greetings to all my fellow hams out there. See you all next time I find something to write about.

73's
Steven Vincent Gunhouse

FREEWAY DRAGSTER

Key Ø starts the race! The roadway moves randomly across the display while you try to avoid other vehicles speeding toward you. Sunday drivers all of them.

Keys 4 and 6 move your dragster in the appropriate direction. The idea is to go as far as you can in the allotted time. Each time you crash, the score is shown. The maximum possible number of miles (kilometers?) is shown on the left compared with how far you have gone, shown on the right.

Crashes cost you 20 units. After the round is over, key Ø restarts a new game.

You may use the page relocatable machine language roadway in your own games. This is located here at 0600.

FREEWAY DRAGSTER LISTING

```
0200  BEGIN: A2CA   VARS   -- I addresses initial variables values
  02         FF65   GET    -- Initialize all variables
  04         22C0   SET    -- Set up display

  06  START: F00A   ;KEY ? -- Wait for starting keypress
  08         3000   ;SK=0  -- Skip if it was key Ø
  0A         1206   START  -- Else go back for next keypress

      ;MOVE BARRIERS

  0C  LOOP:  0600   ;MLS   -- Do sub to move roadway
  0E         FAFB          -- Specifies VA VB for MLS
0210         7B01   ;VB+1  -- Count loops
  12         3B04   ;SK=4  -- If = 4, skip into next part
  14         122E   MOCAR  -- Else go move the dragster
  16         6B00   ;VB=00 -- Reset VB=0
  18         8A54   ;VA+V5 -- Add roadway direction constant
  1A         4A00   ;SK≠0  -- If not at left edge, skip
  1C         1228   COMP   -- Else compliment to reverse travel
  1E         4A1C   ;SK≠1C -- If not at right edge, skip
0220         1228   COMP   -- Else compliment to reverse travel
  22         C007   ;RND   -- V0=random number 0-7
  24         3000   ;SK=0  -- If V0=0 (12.5% of the time) skip
  26         122E   MOCAR  -- Else go move car

      ;2's COMPLIMENT CAR'S VX TO CHANGE
      ;DIRECTION OF ROADWAY TRAVEL

  28  COMP:  60FF   ;V0=FF -- V0 is set to $FF for compliment
  2A         8503   ;XOR   -- Exclusive OR with FF compliment
  2C         7501   ;V5+1  -- Forms binary 2's compliment
```

```
            ;MOVE CAR/SHOW BARRIER

  2E   MOCAR: 0600   ;MLS    -- Do sub to move road
0230          FAFB           -- Specifies VA,VB for MLS
  32          A2DA    CAR 1  -- I addresses bits for dragster
  34          DCDB   ;ERASE  -- Clear old dragster
  36          6004   ;V0=4   -- Test for key 4
  38          E0A1   ;KEY4?  -- Skip on not key 4
  3A          7CFE           -- Adjust VX to go left
  3C          6006   ;V0=6   -- Test for key 6
  3E          E0A1   ;KEY6?  -- Skip on not key 6
0240          7C02   ;VC+2   -- Adjust VX to go right
  42          DCDB   ;SHOW   -- Show new dragster
  44          3F00   ;SK=0   -- Skip if not hits
  46          1274    HIT    -- Else go crash sequence

            ;SEND CARS DOWN

  48          3400   ;FLG=0  -- Skip if no cars being shown
  4A          1252    SEND   -- Else continue moving car down
  4C          6900   ;V9=0   -- Set VY=0 for new cars
  4E          C40F   ;RND    -- Get a random VX coordinate
0250          7407   ;V4+7   --   between 7 and 22 (decimal)
  52   SEND:  8840   ;V8=V4  -- Transfer random # to V8 (car VX)
  54          88A4   ;V8+VA  -- Add VX of road to VX of car
  56          A2E6    CAR2   -- I addresses bits for the car
  58          D898   ;SHOW   -- Show car at V8,V9
  5A          3F00   ;SK=0   -- If not hits, skip
  5C          1274    HIT    -- Else go crash sequence
  5E          D898   ;ERASE  -- Erase the car (to animate)
0260          7902   ;V9+2   -- Increase car's VY coordinate
  62          491E   ;SK≠1E  -- Skip next if not at bottom
  64          6400   ;FLG=0  -- Else signal for a new car

            ;MILEAGE/TIME

  66          7601   ;V6+1   -- Add one to player's score
  68          7701   ;V7+1   -- Add one to maximum score
  6A          37FF   ;SK=FF  -- If at maximum score, skip
  6C          120C    LOOP   -- Else go back to continue

            ;DISPLAY SCORES/END OF GAME

  6E   END:   22A8    SCORE  -- Do sub to display final score
0270          00E0   ;ERASE  -- Clear screen for new game
  72          1200    BEGIN  -- Restart (on key Ø)

            ;SUBROUTINES

  74   HIT:   6004   ;V0=4   -- V0=value for tone
  76          F018   ;TONE   -- Sound tone for length of V0
  78          A2DA    CAR1   -- Set I to bits of dragster
  7A          DCDB   ;ERASE  -- Erase the dragster
  7C          A2EE    CRSH1  -- Set I to bits of first crash pattern
  7E          DCDB   ;SHOW   -- Show in place of dragster
```

```
0280            6009  ;V0=9   -- V0=value for Timer
  82            2324   TIMER  -- Do sub, wait for effect
  84            DCDB  ;ERASE  -- Erase the crash scene
  86            A2FA   CRSH2  -- Set I to crash  pattern #2
  88            DCDB  ;SHOW   -- Show in place of dragster
  8A            6014  ;V0=14  -- Set amount to
  8C            8605  ;V6-V0  --   subtract from player's score
  8E            4F00  ;SK≠0   -- If not negative, skip
0290            6600  ;V6=0   -- 0 is as low as you can go
  92            6040  ;V0=40  -- V0=amount for Timer
  94            2324   TIMER  -- Do sub, wait for effect
  96            22A8   SCORE  -- Do sub.  Display score

      ;RESET/RESTART

  98            8F70  ;VF=V7  -- Save V7,V6 in VF,VE for the
  9A            8E60  ;VE=V6  --   reset (probably poor planning!)
  9C            A2CA   VARS   -- Set I to initial values
  9E            FD65  ;GET    -- Reinitialize V0-VD
02A0            87F0  ;V7=VF  -- Restore V7,V6 scores held
  A2            86E0  ;V6=VE  --   in VF,VE
  A4            22C0   SET    -- Do sub to set up display
  A6            120C   LOOP   -- Go loop for more

  A8  SCORE:    00E0  ;ERASE  -- Clear the display
  AA            6C0A  ;VC=0A  -- VX for display
  AC            6D08  ;VD=08  -- VY for display
  AE            8060  ;V0=V6  -- V0 passes score to sub
02B0            2306   NUMB3  -- Do sub.  Display player score

  B2            6C28  ;VC=28  -- VX moves over (right)
  B4            8070  ;V0=V7  -- V0 passes score to sub
  B6            2306   NUMB3  -- Do sub.  Display maximum score

  B8            60B0  ;V0=B0  -- V0=value for Timer
  BA            2324   TIMER  -- Do sub.  Wait
  BC            00E0  ;ERASE  -- Clear screen again
  BE            00EE  ;RETN

02C0  SET:      A2DA   CAR1   -- Set I to dragster pattern
  C2            DCDB  ;SHOW   -- Display dragster
  C4            0600  ;MLS    -- Do sub to show roadway
  C6            FAFB          --   using VA,VB
  C8            00EE  ;RETN

      ;DATA SECTION

  CA  VARS:     0000  ;V0,V1
  CC            0000  ;V2,V3
  CE            0001  ;V4,V5
02D0            0000  ;V6,V7
  D2            0000  ;V8,V9
  D4            1000  ;VA,VB
  D6            1D15  ;VC,VD
  D8            0000  ;VE,VF
```

```
02DA   CAR1:  10 00 10 44 54 00 38 38 BA BA 82 00

02E6   CAR2:  84 B4 84 30 00 30 B4 84

02EE   CRSH1: 40 00 20 84 54 00 38 B8 BA B2 44 00

02FA   CRSH2: 00 00 40 24 84 50 38 70 39 BA C4 00

0306   NUMB3: A320   C-3DD -- Set I to 3 byte work space
  08          F033  ;CNVRT -- Convert V0 to decimal
  0A          F265  ;GET   -- Let V0,V1,V2=those digits
  0C          F029  ;SET I -- I=bits in ROM for V0 value
  0E          DCD5  ;SHOW  -- Display first digit
0310          7C05  ;VC+5  -- Increase VX
  12          F129  ;SET I -- I=bits in ROM for V1 value
  14          DCD5  ;SHOW  -- Display second digit
  16          7C05  ;VC+5  -- Increase VX
  18          F229  ;SET I -- I=bits in ROM for V2 value
  1A          DCD5  ;SHOW  -- Display second digit
  1C          7CF6  ;VC-A  -- Reset X coordinate
  1E          00EE  ;RETN
0320   C-3DD: 0000         -- Work space
  22          0000

  24   TIMER: F015  ;TI=V0 -- Set CHIP-8 timer=V0
  26   TIME:  F007  ;V0=TI -- Test timer/V0=current time
  28          3000  ;SK=0  -- If=0 skip to end sub
  2A          1326   TIME  -- Else loop to keep checking
  2C          00EE  ;RETN  -- Return from subroutine

       ;MLS ROADWAY SUB

0600          45 A6 45 A7 06 FA 07 AF 06 FA 3F F6 F6 F6 22 52
  10          07 FA 1F FE FE FE F1 AE 9B BE F8 E0 BC F8 00 AC
  20          8F 32 2C 9C F6 BC 8C 76 AC 2F 30 20 F8 02 AF EE
  30          9C F3 5E 1E 8C F3 5E 1E 1E 1E 2F 8F 3A 30 8E FC
  40          18 AE 9E 7C 00 BE FB 10 3A 2C 12 D4 00 00 00 00
```

MULTIPILE NIM
by
Kurt Hafner

This program uses a modified CHIP-8 type interpreter, which is included here. This program will be available on cassette from VIPHCA. The memoru dump is printed here for those of you who can't wait! Load 6 pages. Directions for play are at the end of the program.

```
 Address
0000   C4 91 BB FF 01 B2 B6 F8   CF A2 F8 00 A5 F8 02 B5
0010   F8 81 B1 F8 46 A1 90 B4   F8 1D A4 30 E0 E2 69 96
0020   B7 E2 94 BC 05 F6 F6 F6   F6 32 42 FE FC 45 AC 45
0030   F9 F0 A6 05 F6 F6 F6 F6   F9 F0 A7 4C B3 0C A3 D3
0040   30 1F 45 B3 45 30 3E 00   FA 00 F3 01 83 01 8B 01
0050   75 00 D7 00 DA 01 AE 01   91 01 CA 01 EA 01 D1 00
0060   65 01 9F 01 01 06 FA 07   BE 06 FA 3F F6 F6 F6 22
0070   52 07 FE FE FE F1 AC 9B   BC 9A BF 8A AF 45 FA 0F
0080   AD A7 F8 D0 A6 93 B7 87   32 A3 27 4F BD 9E AE 8E
0090   32 9B 9D F6 BD 97 76 B7   2E 30 8F 9D 56 16 97 56
00A0   16 30 85 00 EC F8 D0 A6   93 A7 8D 32 D0 2D 06 F2
00B0   32 B5 F8 01 A7 46 F3 5C   02 FB 07 32 C9 1C 06 F2
00C0   32 C5 F8 01 A7 06 F3 5C   2C 16 8C FC 08 AC 3B AA
00D0   F8 FF A6 87 56 12 D4 45   56 D4 45 E6 F4 56 D4 00
0E00   9B BF F8 FF AF 94 5F 8F   32 F2 2F 30 E5 23 42 B5
0F00   42 A5 D4 15 85 22 73 95   52 25 45 A5 86 FA 0F B5


0100   D4 45 A3 E6 63 26 D4 98   56 D4 F8 81 BC F8 95 AC
0110   22 DC 12 56 D4 06 B8 D4   06 A8 D4 E6 15 30 EE E6
0120   8A F4 AA 9A 7C 00 BA D4   00 F8 81 BA 06 FA 0F AA
0130   0A AA D4 1A 1A EA F8 FF   AE AF 06 FF 64 1F 33 3B
0140   FC 64 FF 0A 1E 33 42 FC   0A 73 8E 73 8F 5A D4 06
0150   B8 98 3A 51 D4 22 86 52   F8 F0 A7 07 5A 87 F3 17
0160   1A 3A 5B 12 D4 22 86 52   F8 F0 A7 0A 57 87 F3 17
0170   1A 3A 6B 12 D4 45 76 76   33 95 7E 07 3B 84 E6 F7
0180   3B 88 D4 45 E6 F3 3A 82   15 15 D4 45 E6 F3 3A 88
0190   D4 45 07 30 8C 7E 22 87   52 86 A7 33 6B 30 5B 45
01A0   F6 E6 62 26 33 A9 36 88   D4 3E 88 D4 00 00 45 FA
01B0   0F 3A B6 07 56 D4 AF 22   F8 D3 73 8F F9 F0 52 E6
01C0   07 D2 56 F8 FF A6 94 7E   56 D4 45 AA 86 FA 0F BA
01D0   04 19 89 AE 93 BE 99 EE   F4 56 76 E6 F4 B9 56 45
01E0   F2 56 D4 3F E3 37 E5 E6   6B D4 E5 86 FC 01 85 38
01F0   F9 F4 A5 95 7C 00 B5 25   D4 F7 A5 95 7F 00 30 F6
```

4.05.09

MULTIPILE NIM

This is is main body of the program, in conventional CHIP-8 notation. It requires the interpreter printed above and will not function with the regular CHIP-8 interpreter.

```
0200   C703 7703 6801 6A05    621D A501 C00F 50A1
0210   1216 80A5 120E 7001    F81E F055 5781 1224
0220   7801 120A 6118 8B7E    8BBE 81B5 6801 A501
0230   F81E F065 838E 833E    833E 8314 73FE A501
0240   641E D341 7302 A500    6C01 85CE 855E 8420
0250   8455 D341 7C01 5C01    124A 7801 5871 122E
0260   F60A 5671 0188 1260    4600 6601 A501 F61E
0270   F065 4000 1260 836E    833E 833E 8314 73FE
0280   641F A501 D341 F50A    350F 1290 D341 1260
0290   7302 5051 A500 4500    6501 A500 2402 A501
02A0   641F 73FE D341 6C3C    FC4F 00F2 6801 6900
02B0   6D00 A501 F81E F065    4000 12CC 3001 12C6
02C0   8B80 7D01 12CC 8E80    8C00 7901 7801 5871
02D0   12B2 3900 1360 3D00    130C 00E0 6408 630E
02E0   A460 D346 7306 A466    D346 7305 A46C D346
02F0   7306 A472 D346 7305    A478 D346 7306 A47E

0300   D346 FB0A 3B0F 1302    00E0 1200 2440 3D01
0310   134E 6501 86B0 6001    2402 00E0 6408 630D
0320   A484 D346 7307 A48A    D346 7306 A490 D346
0330   730A 00F2 A496 D346    7307 A49C D346 7303
0340   A4A2 D346 FB0A 3B0F    1342 00E0 1200 24B0
0350   6501 6001 6C3C FC4F    2402 2440 1260 00F2
0360   2440 3901 137E 86E0    80C0 6801 8D82 8D83
0370   85C0 85D5 6C3C FC4F    2402 2440 1260 6800
0380   24D0 24B0 3B00 13AA    A501 F81E F065 C50F
0390   7501 5501 139A 8505    1392 6C3C FC4F 2402
03A0   2440 1260 00F2 00F2    00F2 A501 F81E FEE3
03B0   6501 24D0 4B00 13CC    7501 55E1 13B2 7801
03C0   5871 13AA 8875 13AA    00F2 00F2 8680 6C3C
03D0   FC4F 80E0 2402 2440    1260 A606 FD30 33FC
03E0   06FD 343B FC06 FA0F    FEFE FC7C FF93 BFE6
03F0   F89C A646 BE06 FC01    AE56 F8FB A606 5ED4
```

Program continues on next page.

4.05.10

MULTIPILE NIM

```
0400   80F8 6C20 FC4F A500    836E 833E 833E 8314
0410   6801 8B0E 8BBE 8420    84B5 6C3C D341 FC4F
0420   5581 142A 7801 7404    141C A501 F61E 8055
0430   F055 00EE 8055 F055    00EE D1A3 731A 0403
0440   641F 631E 887E 888E    8385 00F2 6939 8835
0450   A500 D341 7301 5831    00EE 1452 0448 1250
0460   F080 80B8 90F0 F090    9090 90F0 F820 2020
0470   2020 F080 8080 80F0    9090 90F0 9090 F191
0480   91F1 9091 8850 2020    2020 F090 9090 90F0
0490   9090 9090 90F0 8888    88A8 D888 8080 8080
04A0   8080 88C8 A898 8888    01A5 A5A2 6A58 0850
04B0   C80F 7801 5871 14BC    8875 14B4 A501 F81E
04C0   F065 4000 14B2 8680    00EE C1A0 4250 8124
04D0   6A01 6B00 A501 FA1E    F065 9A80 8055 8B03
04E0   7A01 5A71 14D4 00EE    A491 002A 2C5E 0839
04F0   91A2 E022 28F0 0108    0021 0080 0000 4000


0500   80F8 0500 0606 0401    4A06 88AB B95B 0200
0510   A08E AA24 5B50 4447    64B2 0106 526D 3112
0520   81A0 AD21 4252 503C    0030 81AD 00CA 1446
0530   00A5 A485 301C 1212    4405 21B3 605B 4EF2
0540   00B8 02C2 0201 4012    0080 E000 7200 0890
0550   C180 3041 4848 0722    AB00 0080 0248 2240
0560   1000 8000 4F2E 8802    A0AE 3000 DA58 0740
0570   A024 8404 4000 4008    0908 00A2 9800 401A
0580   A629 8100 6A73 1216    21A5 8242 787A 121B
0590   01B5 E3AC E71B C058    A4AC 09A4 42DF C759
05A0   64A5 2700 572B 4004    898D B3A5 310A 0011
05B0   EDA2 FC86 CAC4 F252    9787 8258 6A31 5716
05C0   0400 90A5 4018 C81A    2124 8415 1030 0040
05D0   0003 2101 1040 D344    A22B 8020 9264 C248
05E0   C5CC A041 4A48 000A    9D8D 0110 040C 4220
05F0   21A4 0025 42C0 42D2    4584 E600 0049 5010
```

* * * END of PROGRAM * * * *

*Note: there may be some extraneous data in the latter parts of the last page of memory since this listing was produced from a memory dump.

4.05.11

MULTIPILE NIM

THE GAME

Objective:

To force your opponent (the VIP) to take the last marker.

The Play:

1. On a given move the player may remove one or more markers from any chosen column.
2. At least one marker must be removed at each turn.
3. Markers may not be taken from more than one column on a single turn.
4. The maximum number of markers that may be removed in a single move is the total number of markers remaining in the chosen column.
5. Players (you and the VIP) alternate moves.
6. The player taking the last marker loses.

THE PROGRAM

Set the RESET switch to the RUN position to start the game. From three to six columns of markers will appear, each column having from one to six markers. A short line will be present beneath each column of markers. This line will remain throughout the game to indicate the position of the column even after all the markers have been removed.

You will always be given the first move. Begin by pressing the number of the column from which you wish to remove markers. The line beneath the chosen column will widen indicating your choice. (The columnns are numbered from left to right beginning with the left-most column). If you change your mind about your choice of column, press key F. The wide line will disappear and you may select another column.

With the desired column selected, press the key corresponding to the number of markers you wish to remove. The markers will be removed one at a time from the chosen column.

It is now the machine's turn to move. It will first draw a continuous line under all of the columns. The VIP will then ponder its move and remove some number of markers one at a time from the column it chooses. It will then remove the line from beneath the columns to indicate that it is again your turn. If the VIP has been forced to take the last marker, it will concede "YOU WIN". If you have removed the last marker,it will gloat "GOTCHA". When the game is over, press key F to start a new game.

Here are some additional notes about the response of the VIP.

1. If you select an empty or nonexistent column, the VIP will not respond and you must choose again.
2. If you attempt to remove more markers than a column contains, all the markers in that column will be removed.
3. If you attempt to remove no markers from a column by pressing key 0 (an illegal move), one marker will be removed.

The VIP has been programmed to play the best possible game. However you get the first move which gives you the advantage in most cases . But be careful. One mistake and "GOTCHA"!

EDITORIAL

by Tom Swan

I recently read, can't remember exactly where, that IBM Corporation is more likely to hire a music major applicant than a math major. A musician, the rumor goes, deeply appreciates the importance of discipline and is more likely to make a good computer programmer. As I write, I can hear the complaints from the mathematicians among you mixed in with the somewhat smug glow I feel eminating from you musical subscribers.

Having been a professional musician for 13 years, I'm tempted to choose sides, but suspecting IBM's hiring practices to be quite a bit less abstract I'd rather just not take such statements seriously. Certainly there are a lot of mathematicians who make good programmers along with plenty of wailing guitarists to whom "binary" may seem more the malopropistic description of one's sexual preferences than an endearment of the computer professional.

The comment stays with me, however, and brings up a question that often bugs me. What <u>are</u> the ingredients of a good computer programmer? If we can set down, at least in general, the qualities of a professional, then aspiring amateurs will have the advantage of real goals to attain. No one should make absolute statements like that IBM rumor, but we should be able to find some unifying attributes by which good and bad programming, and therefore programmers, may be separated.

Discipline, as suggested by the dubious rumor, <u>is</u> one of those qualities. Without a disciplined meticulousness, a person is not likely to ever become more than a casual programmer. There is a good deal of tedious, painstaking effort that goes into even the simplest of programs, and those persons who get satisfaction from dealing with detail are less likely to become frustrated by the microcosms of microcomputing.

I do not use "discipline" to mean the acculumation of knowledge (e.g. the "discipline of calculus" of the "discipline of music"). To me, discipline is that centrally located eagerness that glues a writer to his desk and his pen to the page forcing phrases out of the unconscious into permanence even -- especially even -- when ideas are more easily left ungerminated. The "discipline of programming" is a redundancy; the "self-discipline of a programmer" is a requisite. The true programmer will program and program and program. If it isn't right, it will be redone until it is. If it isn't understood, it will be reread until clear.

Programmers are fanatically, obsessively in love with curiosity. Their skills are contained in an expanding circle, knowledge inside, mysteries out. The more knowledge we acculumate; the more mysteries we encounter on the edges of what we know. Our curiosity grows with our discoveries. Without the discipline to make those discoveries, we would soon become lost, lacking purpose inside a bubble of facts.

Discipline and a natural curiosity; what else? Intelligence?

To a degree, but having a high IQ does not really mean much. That is more a description of potential, which when combined with other qualities may lead to competence, but could very well remain unfulfilled. I've seen this happen. Look at the musical prodigy who ends as fifth violinist in some obscure city! In fact, the primary requisite, discipline, may come harder to the super intelligent individual who is perhaps too used to attaining intellectual goals with ease. Those of us who are not in that category -- and most of us aren't -- naturally develop good, disciplined work habits because we have to.

Why do I make this point? Because, traditionally, programmers -- even the word "programming" -- has to the general public been associated with the (perhaps fictional anyway) breed of masterminds believed to be in scientific control of the world. Let's dispel that myth right now! Home computing if it is to belong to the people, must not remain the mythical domain of ingeniousness. Every time I hear someone say "Well, what could I possibly do with a computer?" I want to spend the rest of the day answering their skepticism which I interpret as pure and basic interest contained by a fear of failure. We programmers bear the responsibility of relieving those fears and inspiring that interest. In that way we are true pioneers.

Again I sense I may be broiling a controversy. I do not mean to exclude intelligent people from programming. That would be plain dumb. I only mean to criticize the attitude that programming is for the brainy alone. Nothing could be more contrary to the purpose of home computing or to the reasons behind publishing this newsletter. I sincerely hope -- and publicly issue the call here -- that those of you blessed with high mental ability take the lead in bringing programming concepts down to earth. The desire and willingness to share what you know, therefore, is my third proposed general quality for a programmer.

In your quest for the binary grail, you will receive back exactly what you are willing to contribute. That is the lifeblood of the VIPER and the future of personal computer programming.

Discipline. Curiosity. Participation. As a computer programmer, these are your strengths.

*Editor's note:

This Editorial was prepared by Tom some time ago, for possible publication in ARESCO's VIPER. It has been on hand here at VIPHCA HQ for a while also, but nothing of what Tom says has dimmed in the interval. RS

- A Real Time Control Program -

Currently my VIP is running a real-time program for the control of solenoid actuated sprinkler valves in my parents garden. Utilizing the effective 60 hz interupt of the 1861 video chip, the machine language program can maintain an accurate clock. On each interupt cycle the program makes a series of calls to a single efficient countdown count-up subroutine referencing a list of constants and updating a list of variables that include: sixxths of a second, tenths of a second, seconds, minutes, hours, day of the week, day of the month, month, and year. The program also maintains two countdown clocks. The first is used to keep track of how long to run a circuit. The other is reserved for future use. Each time the clock is updated it compares the time against a task list stored in memory - the sprinkler schedule. When the times are equal the sprinkler on-time is loaded into the countdown clock. The sprinkler circuit number is output to the parallel port as a single 4 bit hex digit. After passing through some simple transistor drivers, the bits drive effectively 4 relays whose switch contacts are wired to form a binary tree. All relays at rest is not used and therefore allows for 15 circuits. The first relay is a double throw single pole. The second is a double pole double throw. The third is a four pole double throw. The fourth and last relay is an eight pole double throw producing sixteen outputs for a common wired 24Vac sprinkler system. A disadvantage is that only one circuit can operate at one time, but for my application this is fine.

Although I refer to the clock as a program, it is actually an interrupt subroutine. Every sixty times a second this program does all of the above, leaving the processor free to execute maintenance programs and display the real time variables (i.e. the correct time). Most of the time the real time routine execution is very short, such as when the sixths of a second counter is incremented and is not reset. In this case there is no point in executing further so the routine merly exits. If the sixths counter were to be reset, this would indicate that the tenths of a second counter should be incremented. If the tenths were reset, this would indicate a change in seconds. It would then be clear that the longest execution time for the real time routine would be at midnight on December 31st, 1999. Yet, even then, the routine would not use up all of the allowable machine cycles between interrupts.

As the program stands right now there are no maintenance routines. In fact the processor waits in an endless loop between interrupts. The clock program is complete, yet, the sprinkler routines include only three: one indicates date and time to start, the second indicates how long to run a particular circuit, and the final one marks the end of the schedule. As it is, the program can reside in less than 3K of ram. When it is completed it will probably use all 4K, including the display screen area.

At the rate I work on things,program,(completion) listings and documentation should be ready for publication near the end of the year.

Jim Gard, (Cromemco computer repair technician)
Home Address - 5469 Beechwood Lane, Los Altos, CA 94022

4.05.16

ANGELS WE HAVE HEARD ON HIGH

Step 1: Load the PIN-8 interpreter.

Step 2: Load the following:

```
0259    FF
02E0    0303 0303 0404 0404 0505 0505 0606 0606
0300    0105 080C 0105 080C 0F14 191E 2125 0F14
0310    191E 2125 2700 0000 0000 0000 0000 0000
0320-037F   0000
0380    0105 080C 0105 080C 0F12 1518 1B1F 0F12
0390    1518 1B1F 2100 0000 0000 0000 0000 0000
03A0-03FF   0000
0400    006F 6F6F 7292 30AF 6F6D 6F72 8F2D ABB2
0410    3432 302F B032 302F 2DAF 302F 2D2B 8D26
0420    A66B 6D6F 70AF ADEB 0000 0000 0000 0000
0430-04FF   0000
0500    006B 6B6B 6F6D 6AAB 6B6A 6B6B 6A66 A66F
0510    6DAC 6D6B AA6B 6AA8 6A68 A668 6A6B 6BAB
0520    AAEB 0000 0000 0000 0000 0000 0000 0000
0530-05FF   0000
0600-06FE   0000
06FF    ED
```

Break Table:

```
0270    1281 E016 81E0 FE12 81E0 1681 E0FE 1281
0280    E016 81E0 FF00 0000
```

Step 3: Store on tape 7 pages.

4.05.17

HAHVAH NAHGEELAH

Step 1: Load the PIN-8 interpreter.

Step 2: Load the following:

```
0259   FF
02E0   0303 0303 0404 0404 0505 0505 0606 0606
0300   0107 0D13 0107 0D18 1D23 2913 1D23 2918
0310   2F31 3535 3E3E 474F 5600 0000 0000 0000
0320-037F  0000
0380   0101 0101 0101 0101 0101 0101 0101 0101
0390   0204 0808 1111 1A22 0100 0000 0000 0000
03A0-03FF  0000
0400   0066 6620 2A27 266A 6A20 2D2B 2A6B 6B20
0410   2E2D 2B6A 0706 27AA 6A07 0627 A62A 6A27
0420   2626 6627 6726 2424 6464 4706 2424 6BAB
0430   AB6B 6B6B 6B0B 0B2B 4E0D 2B2E 2D2B 0D0D
0440   2D50 0E2D 302E 2D0D 0D2D 720D 0D2D 7226
0450   260E 0D0B 0AAB E000 0000 0000 0000 0000
0460-04FF  0000
0500   00E0 A2A2 6262 6262 0202 2246 0422 2624
0510   2204 0424 4706 2427 2624 0404 2466 0404
0520   2466 2626 4604 A200 0000 0000 0000 0000
0530-05FF  0000
0600-06FE  0000
06FF   ED
```

Break Table:

```
0270   1201 E016 01E0 0000 0000
```

Step 3: Store on tape 7 pages.

4.05.18

VIP Hobby Computer Assn.
32 Ainsworth Avenue
East Brunswick, NJ 08816

First
CLASS

A Final Word:

April 1st. will be here before we know it, so I thought that I'd take this early opportunity to let you know that there will be only one more issue of VIPER for the current membership year. In the next VIPER you will be asked to renew your membership. In all probability, dues will remain at $12 per year. But we may have to consider an increase in future years. Postage costs have stayed about the same over the past two years, but printing costs have gone up a little. (We are, however, getting a break on the price since we are long time customers of our printer, The Word Center, in New York City) And once again, I'd like to invite any of you who have written programs to send them in to VIPER. There is still material here on file, but we can always use more. And we can always use short letters about projets you are working on, or a nice hardware or software trick you have discovered.

And Don't forget that the programs in this VIPER are available on cassette for $2.00, payable to VIPHCA. Be sure to mention which VIPER you want on tape. But please plenty of time in case I get swamped!

All the best wishes for an enjoyable and rewarding 1983!

4.05.19