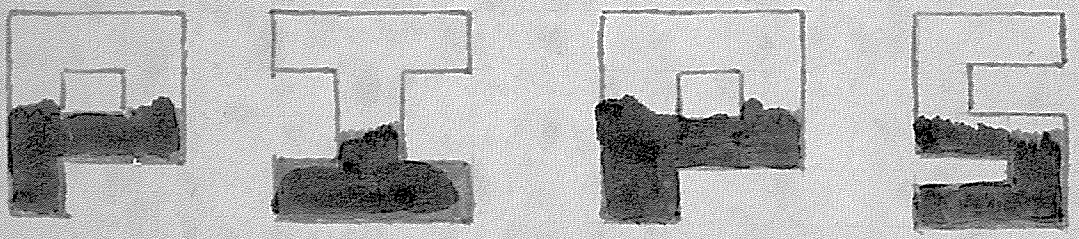
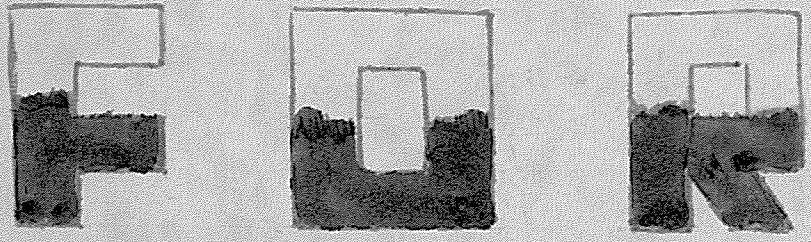
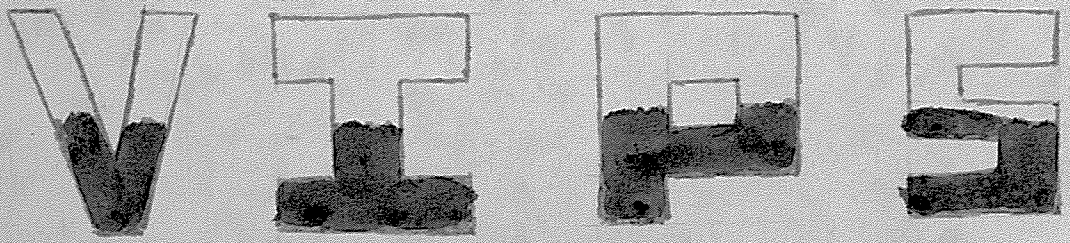






by TOM SWAN

VIPER

The only comprehensive monthly newsletter devoted exclusively to

THE RCA COSMAC VIP

PRESENTS

PIPS FOR VIPS

by TOM SWAN

ARESCO
P.O. Box 1142
Columbia, MD 21044
THE PAPER - VIPER - RAINBOW - SOURCE

P I P S F O R V I P S

Designed And Written By

TOM SWAN

Edited by Terry Laudereau

Published by ARESCO
BOX 1142
COLUMBIA
MD 21044

TO ANNE

EDITORIAL

Those of you who know me from the VIPER will recognize that I have a hard time keeping my "two-cents worth" out of the material I edit - and this book proves no exception. I'm so impressed by Tom Swan's work that I'm inclined to jump up and down, flap my arms around, and holler at you about how good it is. (Since you're reading this "editorial" however, you've already purchased the book, and there isn't anything I could tell you that you won't find out by yourself in very short order!)

I think PIPs For VIPs will prove to be a "classic" in its field. Not only does Tom tell us what the program does and how we can make it perform, but he goes to some length to let us see how the program works, what we can do to modify it, how to do those modifications (and what to not modify!), and which modifications occurred to him as he was writing the program. He tells us what each CHIP-8 variable is used for; what each 1802 register is used for; where all the important subroutines begin; and even gives us checksum data on the CHIP-8 programs, so we can verify our program entry. What more could we ask? This is truly a model of "ideal" documentation! As most of us are sadly aware, even the best of programs available on the market today are deficient in the documentation department. Perhaps other authors will take their cue from Tom Swan's examples here.

I also think PIPs For VIPs can serve as a sort of "laboratory" for people who are trying to learn CHIP-8. Or 1802 machine code. Or how to use machine language subroutines in CHIP-8 programs. When combined with the wealth of material provided by RCA in the 1802 Instruction Manual, the VIP Instruction Manual, and the VIP User Guide, PIPs For VIPs provides a learning tool that is unbeatable in any other language for any microprocessor!

When all this material is supplemented by user interaction in the pages of the VIPER, I predict that the 1802 - and especially the VIP - will not long remain hidden from the public view.

And now for a word from our sponsor (if you don't like commercials, this is the part to skip).

ARESCO offers the tape containing all the programs in this book, for $5.00. If you haven't purchased the tape, you'll have to enter all the code yourself, and there's an awful lot of code here! I advise you to buy the tape; you'll spend more time enjoying and using the programs and less time trying to debug your data entry.

ARESCO also offers the VIPER (you'll find a subscription blank near the end of the book), to which Tom refers in several of the programs here. Now, it isn't essential to your health and happiness that you buy the VIPER - but it will make your VIPping a lot more fun! User interaction - the realization that there are others out there somewhere who are learning and working and dreaming up new ideas for <u>their</u> VIPs, just as you are learning and working and dreaming up new ideas for yours - is a great part of the fun of personal computers. The VIPER is <u>not</u> written by the editors. It is composed solely of reader contributions - programs, harware mods, tutorials, and ideas written <u>by</u> VIP users <u>for</u> VIP users. And you're invited to participate in all this!

Okay - commercial's over. Now it's time to get on with the fun - with some <u>P</u>retty <u>I</u>mpressive <u>P</u>rograms for VIPs!

Terry L. Laudereau
May, 1979

INTRODUCTION

As Editor of the VIPER, the newsletter for owners of the RCA COSMAC VIP, I have been constantly surprised and pleased at the ingenuity and creativity displayed by VIP owners. The VIP was originally designed to allow its owners to create programs utilizing the excellent video display capabilities of the 1802 microprocessor, coupled with the 1861 video display generator. This simple yet elegant combination has provided hundreds of hours of fun and education for thousands of VIP purchasers. The excellent set of games and displays available in the VIP Instruction Manual provided the springboard for dozens of programs we have published in the VIPER.

Although the VIP was never intended to display more than a few alphabetic characters (mostly messages incidental to some game program), there has been a continuing interest on the part of VIP users in displaying text on the VIP. A first step in this direction was the publication in an early issue of the VIPER of a higher resolution interrupt subroutine, permitting twice as many lines of text on a single screen. Later articles doubled this resolution again, and this is the display format used in several of the programs in this volume. This resolution, 128 vertical elements by 64 horizontal elements, allows up to sixteen lines of sixteen characters each.

In part, this book is an exploration of the text capabilities of the VIP. The Character Designer program, coupled with the Messager routines, allows easy display of text messages under CHIP-8 control, while giving the programmer a virtually unlimited capability to design his own character sets. These capabilities are elegantly displayed in the Text Editor 21 and the Disassembler 7 programs, which give the VIP capabilities its designers probably never imagined. In addition, Tom has given us two entertainment programs: Surround and Space Wars. They are among the best games ever designed for the VIP. And the

book is rounded out by a good utility program, the CHIP-8 Editor, which you will find useful in loading and examining the other programs in the book.

Although the programs are valuable in themselves, the wealth of information Tom has presented with each program listing is a "graduate course" in VIP programming. Careful reading of the programs should provide many new insightes into the use of CHIP-8 and machine language on the VIP. The exhaustive documentation should allow the reader to modify any of the programs to meet his special needs or desires. Several of the programs will be even more valuable if interfaced to an external keyboard, such as the VP-600 ASCII Keyboard provided by RCA. Users who make such modifications are invited to submit them to the VIPER for publication.

All VIP owners are indebted to Tom for opening a new application area for the VIP. We look forward to his future efforts.

Richard Simpson
May, 1979

TABLE OF CONTENTS

The information presented in "PIPs For VIPs" is believed to be accurate and reliable. However, no responsibility is assumed by ARESCO for its use; nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of ARESCO.

TEXT EDITOR 21

T E X T E D I T O R 2 1

INTRODUCTION

Running in 3K or 4K of memory, TEXT EDITOR 21 is a COSMAC VIP's answer to the text editing capabilities of the "big boys". There are video boards on the market costing more than your computer - that don't have the features of this simple piece of software.

TEXT EDITOR 21 gives you auto scrolling; up to six pages, sixteen lines by sixteen characters; underline cursor, and tape read/write routines so you don't need to reset the computer to record and play back data.

Characters are normally entered from left to right, with your cursor performing an automatic carriage return at the end of the lines. Typing past the last line on any page will trigger a scroll up operation - additional text is then typed in on the bottom line. Any page may be shown in any order. Even reverse video, black on white, is provided. You can turn off the cursor. You have over 1500 characters of storage in a 4K system, including lower case letters. Practically the whole ASCII set is here! (The only missing characters are the "@" and the "&" - because I could not construct them within a 3 x 5 area.)

So why the name "TEXT EDITOR 21"? Because that's how many functions you have at your command! Here they are:

KEY	FUNCTION
8	Cursor left
9	Cursor right
A	Scroll up
B	Scroll down
C	Control Select (not a function by itself)
D	Carriage return
E	Cursor up
F	Cursor down

.....but just wait. That's not all. Remember key C?

After pressing Key C, you have a few more functions at your fingertips! Just look at this array:

KEY	FUNCTION
C/0	Escape control
C/1	Page backward
C/2	Page forward
C/3	Show page "N"
C/4	Cursor on/off (underline visible or invisible)
C/5	Reverse field video (black on white or white on black)
C/6	Insert line
C/7	Close up (delete) line
C/8	Available for expansion
C/9	Available for expansion
C/A	Erase text buffer
C/B	Tape read
C/C	Home cursor
C/D	Erase to end of line
C/E	Erase to end of page
C/F	Tape write

(I'm almost out of breath, but also almost finished.)

In addition to these functions, you have two keys available for future expansion (such as insert/delete characters). Even though TEXT EDITOR 21 is small (the program requires less than 4 pages of memory), there are over 80 bytes of memory available for adding your own custom routines. How about "OUTPUT TO PRINTER", or "INSERT/DELETE CHARACTER", or "ERASE TO END OF TEXT" ("NEW"?), or "REPEAT CHARACTER", or "SET TAB SPACE"? After reading the program description, you should be able to create almost any routine you want. TEXT EDITOR 21 is written using the standard Call and Return subroutine techniques, and all functions are selected by jump tables. New routines go in as easy as 01, 10, 11. (That's 1, 2, 3, in binary.) And, if you don't like my choice of keys, I won't be insulted - go ahead and change them. Deciding which key does what is even easier than adding new routines; in fact, I had to stop myself from rearranging things every few minutes, since a key switch requires changing only one or two bytes.

How are characters entered? By typing in their two-digit ASCII code on the hex keypad. At first, I thought this might be far too slow or clumsy to operate, and I'd be forced to break down and buy that zillion key rollover, auto-everything, two shot fuel injected George Risk job I had been drooling over for a month. But not so. In an hour I knew most of the codes for the capital letters, and now I have to look up only the codes for the most obscure punctuation symbols. (Since typing over any character simply changes the character to what you want, mistakes are easy to correct.) It's not any more difficult than learning Morse Code, and after you've learned it, you can write labels in programs where you need to figure ASCII strings without taking half an hour - or more - to look it all up in an ASCII Code Reference Chart.

In the following program description (and in the companion DISASSEMBLER 7 that follows), I have attempted to be as descriptive as possible of each and every routine; what it does, how, and why. When I do buy that keyboard - or if you already have one - I want the interfacing job to be quick and easy. Changing from hex to the "real thing" will only take a minute, and with the detail to follow, you should be able to write the I/O routines with a minimum amount of frustration (and a little effort. Very little.).

OPERATION

Due to the elimination of the tone every time a key is pressed, the hex keyboard will have a much different feel than it does in most other programs. With so many functions, your keyboard gets quite a work out and if the tone had been included, beep-a-mania would surely have been the result. Because of this, each

key is sensed only after it is released. Usually a very light tap is all that is necessary, but remember, a key will do nothing until it is released.

The ASCII character set is referenced by the hex codes 00 through 7F (128 codes). Pressing a key less than 8 for the first digit will cause the program to wait for a second digit to complete the code. The selected character will then be entered at the cursor, and a cursor right, carriage return or auto scroll will follow, whichever is appropriate. Pressing keys 8 through F will select one of the first level functions in the part one of the list. These have obvious uses, and playing around with them will describe them better than I can here.

Key C selects the bank of second level functions. A warning tone comes on after "C" is pressed to allow you to press key 0, escape control if you wish to simply continue typing.

Paging backwards and forwards is selected by keys 1 and 2. In all cases page continuity is maintained so that it is impossible to page into the middle of a page. Using these functions insures that the display will be from the beginning of a page. Trying to page forwards past the highest page of text will cause an automatic escape from control. Trying to page back too far will cause the first page to be redisplayed, and also escape control.

Show page "N" allows jumping around memory at will. Press C, then 3, then the number of the page (1-6) desired. This page will then be shown. Pressing key 0, or a key too large will not be accepted, though the display will not necessarily be from the exact same point as it was before.

Cursor on/off allows the use of an invisible cursor when designing graphics or working with text that is interfered with by the cursor. (Certain lower case letters are chopped off -- temporarily -- when the cursor passes under them, for instance) Reselecting this routine will then reset the cursor back to an underline.

Reverse field video. Just what it implies. Black on white, or white on black text is possible at the touch of a (2 actually) button. Using black on white during the day and white on black at night is very easy on the eyes.

Insert/close-up line. Position the cursor anywhere on the line you wish to delete or anywhere on the line you wish to move down, leaving a space to enter a new line. Practice these before using them on an important document. In all cases the entire line will be affected regardless of the position of the cursor. All text is moved up or down below that position with old lines at the end of text shifted out of the buffer when an insert line function is selected. DO NOT INSERT LINES AT THE LAST LINE OF TEXT. Nothing really drastic can

happen, but the program needs at least two lines of text for inserting a line. It will therefore use the next to last line, moving it down one. (This is minor. You will have no reason to use this function on the last line as inserting a line here wuuld result in its loss anyway.)

Erase text buffer sets all memory to ASCII spaces (20's). The cursor is automatically homed on page one to make starting new text easy and fast.

Home cursor sends the cursor to the upper left corner of whatever page it is on.

Erase to end of line. The cursor position is unaffected allowing easy changes to portions of text.

Erase to end of page. Same as erase line with the cursor returning to its original position when done. (Demonstrate this to your friends. It's guaranteed to bring "ooo's" and "ahh's" and "oh that's cute." Funny how computers can do that.)

Tape read/write. The ROM routines are used here with the difference that the number of pages recorded or read in is fixed at six. Therefore the read/write operation begins immediately after pressing keys B or F. For recording, start the tape <u>before</u> pressing key F. By fooling around a little with the program counter, I was able to cause a return to the program rather than the static display reached with the normal operating

system following a tape operation. This means you should never have to reset the run switch after the initial power up as the first page of text will automatically be displayed, ready for your commands, following either taping operation.

Backing up to the first level functions again, please note that moving the cursor right past a line end causes an auto carriage return, so typing does not need to be interrupted. Moving left will cause the cursor to eventually stop at the far left. Going up or down too far causes an automatic home and tone warning. Scrolling up does not affect the display cursor position to allow typing new lines on the bottom of pages as old lines are scrolled up automatically out out the way. Scrolling back homes the cursor. (Presumably you are scrolling back to get to a new top line)

In addition to the program itself, you need to enter your character set in the two pages just below the four used for display refresh. This character set conforms to the format of my program "A CHARACTER DESIGNER for theVIP" so it will not be discussed here. (see program description for further details, however.) This also applies to DISASSEMBLER-7.

Systems with only 3K of memory can use TEXT EDITOR-21

by carefully following the provided listing. All bytes that need to be changed have been marked, like this:

EXAMPLE: 014C 05/09 3K=05/4K=09

Enter whichever number applies to your system.

REGISTER ASSIGNMENT

R0 DMA pointer
R1 Interrupt PC - for display refresh routine @ 0300
R2 Stack pointer @ 00FF
R3 Normal PC
R4 PC for Call routine @ 0312 - dedicated
R5 PC for Return routine @ 0322 - dedicated
R6 Pointer to Return and arguments to be passed
R7 Display cursor address pointer
R8 Timer in interrupt (Decrementing) R8.0 only. R8.1 holds key presses and ASCII codes
R9 Pointer to first byte data page - initialized to 0400
RA Cursor to data (displacement from R9)
RB RB.1=Display page start address. RB.0 is available
RC PC for ROM keyboard scan. Not dedicated except between successive key presses.
RD Utility
RE Utility - loop counter; RE.1 passes unpacked bit rows in display subroutine
RF Altered in key scan. Utility for other uses; Bit pattern pointer, etc.

MAIN PROGRAM

0000-0027 Initialization
0028-0049 Main loop - Character entry
0050-0096 Tape read/write subroutine
0097-00B2 Close up line subroutine
00B3-00DF 45 bytes available for expansion

RESERVED MEMORY

00E0-00FF Stack - 32 bytes deep
0400-09FF Text area; 6 on-screen pages (0400-05FF for 3K systems - 2 pages)
0A00-0BFF Full ASCII Character set including lower case letters (for 3K systems, 0600-07FF)
0C00-0FFF Display refresh (0800-0BFF in 3K systems)

SUBROUTINE LIST

0100-01FF	Jump table for Control C functions
0110-0116	Control C function decode
0117-0127	Reverse field video
0128-0136	Page forward
0137-0141	Page backward
0142-0143	FILLER
0144-016B	Insert line
016C-0176	Delete and close up line
0177-0197	Erase to end of line
0198-01CE	Erase to end of page
01CF-01E1	Erase text buffer
01E2-01E5	Call to tape read/write routines
01E6-01EF	Call to Cursor On/off routine
01F0-01F4	Call to show page "N"
01F5-01F9	Home cursor (on command)
01FA-01FF	AVAILABLE
0200-020F	Function decode for keys 8-F
0210-0243	Scroll forward. Cursor position unchanged. Error tone @ 023B
0244-025E	Scroll backward - cursor homed
025F-027E	Cursor down
027F-0297	Cursor up
0298-02C5	Carriage return
02C6-02DD	Cursor right
02DE-02EF	Cursor left
02F0-02F6	Control C select
02F7-02FF	AVAILABLE
0300-0311	4-page interrupt with timer
0312-0321	Call routine - runs in R4
0322-032E	Return routine - runs in R5
032F-0352	Display memory page. One page displayed. Cursor off on return
0353-035B	Home cursor. Cursor must be off on entry
035C-035E	Display cursor (also erases cursor, if re-called)
035F-0387	Display character. Displays ASCII character in R8.1
0388-039E	Display bit row. Does one of 8 bit row patterns for a character
039F-03AD	New character at cursor; for entering new characters
03AE-03BF	Erase display pages. Sets 4 refresh pages = 0
03C0-03D2	Cursor on/off. Complements cursor for invisible or underline display
03D3-03DF	Show page "N". Enter # of page to display. 1-6-4K 1-2-3K
03E0-03FF	AVAILABLE

INITIALIZATION

```
0000   91   GHI  R1    ;R1 points to last RAM page
  01   FF   SMI        ;Subtract 3 to point to
  02   03              ;Last four pages
  03   BB   PHI  RB    ;RB is display page pointer
  04   F8   LDI
  05   03
  06   B1   PHI  R1    ;R1.1 = 03
  07   B4   PHI  R4    ;R4.1 = 03
  08   B5   PHI  R5    ;R5.1 = 03
  09   F8   LDI
  0A   FF
  0B   A2   PLO  R2    ;R2 = 00  -Stack (see ML 0016)
  0C   F8   LDI
  0D   02
  0E   A1   PLO  R1    ;R1 = 0302  -Interrupt Routine
  0F   F8   LDI
0010   13
  11   A4   PLO  R4    ;R4 = 0313  -Call Routine
  12   F8   LDI
  13   23
  14   A5   PLO  R5    ;R5 = 0323  -Return Routine
  15   90   GHI  R0    ;(00) R0 still PC
  16   B2   PHI  R2    ;R2 = 00FF  -Stack Pointer
  17   A9   PLO  R9    ;R9.0 = 00
  18   B3   PHI  R3    ;R3.1 = 00  -prepare for use as PC
  19   F8   LDI
  1A   04
  1B   B9   PHI  R9    ;R9 = 0400  -Data Page Pointer
  1C   F8   LDI        ;Load address into
  1D   20              ;R3 to prepare for use as PC
  1E   A3   PLO  R3
  1F   D3   SEP  R3    ;R3 is PC/R0 free for DMA
0020   E2   SEX  2     ;X = 2
  21   69   INP        ;Video on
  22   D4   SEP  R4
  23   03              ;Call Display Memory Page
  24   2F
  25   D4   SEP  R4
  26   03              ;Call Home Cursor
  27   53
```

MAIN LOOP - CHARACTER ENTRY

```
0028   F8   LDI        ;Begin Main Loop
  29   81
  2A   BC   PHI  RC
  2B   F8   LDI
```

```
002C   95
  2D   AC    PLO  RC    ;RC initialized for keyboard scan
  2E   DC    SEP  RC    ;Get keyboard input
  2F   B8    PHI  R8    ;Put first key press in R8.1
0030   F8    LDI
  31   07
  32   F7    SM         ;D - M(R(x)) -Test if key press ≥ 8
  33   33    BPZ        ;Branch if positive or zero-key was 00-07- DF=01 (<8)
  34   3A
  35   D4    SEP  R4
  36   02               ;Call Function Decode (ASCII code not selected)
  37   00
  38   30    BN         ;On Return From Function, branch for next
  39   28               ;Key press cycle  (RC possibly changed)
  3A   DC    SEP  RC    ;Get second keyboard input
  3B   98    GHI  R8
  3C   FE    SHL
  3D   FE    SHL        ;Shift first key press left
  3E   FE    SHL         to move digit to MSB's position
  3F   FE    SHL
0040   F1    OR         ;OR with second digit on stack (D= ASCII code)
  41   B8    PHI  R8    ;R8 contains the new ASCII code
  42   D4    SEP  R4
  43   03               ;Call Display New-Erase Old character @ cursor
  44   9F
  45   D4    SEP  R4
  46   02               ;Call Cursor Right (after displaying
  47   C6               ;A character)
  48   30    BN         ;Branch to 0028 - loop for next command
  49   28               ;End of Main Loop

004A-4F  Filler

                          TAPE READ/WRITE

0050   E2    SEX  2     ;X = 2
  51   96    GHI  R6
  52   73    STXD       ;Push R6.1
  53   86    GLO  R6
  54   73    STXD       ;Push R6.0  (save old R6)
  55   F8    LDI
  56   04
  57   B6    PHI  R6
  58   F8    LDI        ;Set R6 to point to
  59   00
  5A   A6    PLO  R6    ;Text Buffer beginning
  5B   F8    LDI
  5C   02/06             ;(Change for 4K system) (3K=2/4K=6)
  5D   AE    PLO  RE    ;# pages for data transfer
```

```
005E   22    DEC   R2    ;Decrement stack to counter unwanted INC next
  5F   61    OUT         ;Turn off video  (R(x)+1)
0060   83    GLO   R3
  61   A4    PLO   R4
  62   24    DEC   R4    ;Swap program counters - R4 now is PC
  63   93    GHI   R3     and must later be reset to 0313 for Call Routine
  64   B4    PHI   R4
  65   D4    SEP   R4
  66   F8    LDI
  67   80
  68   B3    PHI   R3    ;Set R3.1 to ROM Read/Write Routine addresses
  69   8F    GLO   RF
  6A   FB    XRI         ;Test RF.0 to see if Read/Write selected
  6B   0F                ;(Key press is still in RF.0)
  6C   3A    BNZ         ;Branch to Read Routine
  6D   74
  6E   F8    LDI         ;Begin Call to Tape Write
  6F   91
0070   A3    PLO   R3    ;Set R3 = Tape Write
  71   D3    SEP   R3    ;Do sub - Write RE.0 pages to tape
  72   30    BN          ;Returns via D4 to here
  73   78                ;Branch to exit
  74   F8    LDI         ;Begin Call to Tape Read
  75   C2
  76   A3    PLO   R3    ;R3 points to Read Routine in ROM
  77   D3    SEP   R3    ;Call Tape Read - returns via D4 to next line
  78   E2    SEX   2     ;X = 2                          (video is on)
  79   60    IRX         ;Point to saved data
  7A   72    LDXA        ;Pop R6.0
  7B   A6    PLO   R6    ;Restore R6.0
  7C   F0    LDX         ;Pop R6.1
  7D   B6    PHI   R6    ;Restore R6.1  (R4 still PC)
  7E   84    GLO   R4
  7F   A3    PLO   R3
0080   23    DEC   R3    ;Swap PC's
  81   94    GHI   R4
  82   B3    PHI   R3
  83   D3    SEP   R3    ;R3 now is PC again
  84   F8    LDI
  85   03
  86   B4    PHI   R4
  87   F8    LDI         ;Reset R4 for use
  88   13
  89   A4    PLO   R4    ;As dedicated Call Routine pointer
  8A   F8    LDI
  8B   04
  8C   B9    PHI   R9    ;Reinitialize R9 -Text Data pointer
  8D   F8    LDI
  8E   00                ;To point to first page of text
  8F   A9    PLO   R9
```

```
0090    D4    SEP   R4
  91    03                ;Call Display Memory Page
  92    2F
  93    D4    SEP   R4
  94    03                ;Call Home Cursor
  95    53
  96    D5    SEP   R5    ;Return
```

CLOSE UP LINE

```
0097    D4    SEP   R4
  98    02                ;Call Left Cursor
  99    DE
  9A    8A    GLO   RA    ;Move Cursor left
  9B    FA    ANI
  9C    0F                ;Until @ extreme left (RA.0 = X0)
  9D    3A    BNZ
  9E    97                ;Then procede
  9F    8A    GLO   RA
00A0    FC    ADI
  A1    10
  A2    AF    PLO   RF    ;Set RF to row
  A3    9A    GHI   RA
  A4    7C    ADCI        ;Directly below cursor row
  A5    00
  A6    BF    PHI   RF
  A7    EF    SEX   F     ;X = F
  A8    9F    GHI   RF
  A9    FB    XRI
  AA    06/0A             ;When RF goes past  (3K=06/4K=0A)
  AB    32    BZ          ;Last byte of text
  AC    B2                ;Branch to exit
  AD    72    LDXA        ;Get character below (advance pointer)
  AE    5A    STR   RA    ;Put character above
  AF    1A    INC   RA    ;Next character position
00B0    30    BN          ;Loop to end of text
  B1    A8
  B2    D5    SEP   R5    ;Return
```

JUMP TABLE FOR CONTROL C FUNCTIONS

```
0100    16  Escape Control
  01    37  Page Backwards
  02    28  Page Forwards
  03    F0  Show Page "N"
  04    E6  Cursor On/Off
  05    17  Reverse Field Video
  06    44  Insert Line
  07    6C  Close Up Line
```

```
0108   16   None
  09   16   None
  0A   CF   Erase Text Buffer
  0B   E2   Tape Read
  0C   F5   Home Cursor
  0D   77   Delete to End of Line
  0E   98   Delete to End of Page
  0F   E2   Tape Write
```

CONTROL (C) FUNCTION DECODE

```
0110   DC   SEP  RC   ;Do sub - Key Scan in ROM
  11   AF   PLO  RF   ;Put the key press into RF.0
  12   93   GHI  R3   ;Load RF.1 with page address of Jump Table
  13   BF   PHI  RF   ;RF indexes the above table
  14   0F   LDN  RF   ;Get the byte from table
  15   A3   PLO  R3   ;Put in R3.0 to effect jump to that function
  16   D5   SEP  R5   ;Return instruction for disabled functions
```

REVERSE FIELD VIDEO

```
0117   F8   LDI
  18   03
  19   BE   PHI  RE
  1A   F8   LDI
  1B   B8
  1C   AE   PLO  RE   ;RE points to Erase Routine Memory byte @ 03B8
  1D   0E   LDN  RE   ;Get the current erase byte
  1E   FB   XRI
  1F   FF             ;Compliment
0120   5E   STR  RE   ;Put new byte back
  21   D4   SEP  R4   ;(Routines branch to here that require the next sub)
  22   03             ;Call Display Memory Page
  23   2F
  24   D4   SEP  R4
  25   03             ;Call Home Cursor
  26   53
  27   D5   SEP  R5   ;Return
```

PAGE FORWARDS

```
0128   99   GHI  R9
  29   FB   XRI      ;Test if R9.1 is at last page of text
  2A   05/09         ;(3K=05/4K=09)
  2B   3A   BNZ      ;If not, continue
  2C   2E
  2D   D5   SEP  R5  ;Else return
  2E   99   GHI  R9
  2F   FC   ADI      ;Add 01 to R9.1 for next page
```

```
0130    01
  31    B9    PHI   R9
  32    F8    LDI         ;Then set R9.0 = 00 so not to
  33    00
  34    A9    PLO   R9    ;Accidentally display half pages
  35    30    BN          ;Branch to exit point displaying memory
  36    21                ;And homing cursor @ 0121

                              PAGE BACKWARDS

0137    99    GHI   R9
  38    FB    XRI         ;Test if R9.1 points to first page of text
  39    04
  3A    32    BZ          ;If so, branch to set R9.0 = 00 so to
  3B    32                ;Not page back past first page
  3C    29    DEC   R9    ;Else decrement R9
  3D    89    GLO   R9
  3E    3A    BNZ         ;Until R9.0 = 00 (R9.1, then, is one
  3F    3C                ;Less than before unless displaying
0140    30    BN          ;From mid page)
  41    21                ;Then branch to exit point

0142-43  Filler

                              INSERT LINE

0144    E2    SEX   2     ;X = 2
  45    D4    SEP   R4
  46    02                ;Call Carriage Return
  47    98
  48    D4    SEP   R4
  49    02                ;Call Cursor Up - cursor will now be at
  4A    7F                ;Extreme left and cannot be on last line
  4B    F8    LDI
  4C    05/09             ;3K=05/4K=09
  4D    BE    PHI   RE
  4E    BF    PHI   RF    ;RE.1, RF.1 = Last memory page of text
  4F    F8    LDI
0150    EF
  51    AE    PLO   RE    ;RE = 05/09 -EF- Last character next to last row
  52    F8    LDI
  53    FF
  54    AF    PLO   RF    ;RF = 05/09 -FF- Last character last row
  55    0E    LDN   RE    ;Get data above
  56    5F    STR   RF    ;Put data below
  57    9E    GHI   RE
  58    52    STR   R2    ;Push RE.1
  59    9A    GHI   RA    ;Test if RE.1 = RA.1 (line pointer)
  5A    F3    XOR
```

```
015B   3A    BNZ        ;If not equal, continue
  5C   63
  5D   8E    GLO  RE    ;Test if RE.0 = RA.0
  5E   52    STR  R2
  5F   8A    GLO  RA
0160   F3    XOR
  61   32    BZ         ;If equal, branch to exit
  62   67
  63   2E    DEC  RE    ;Else decrement both pointers for next data
  64   2F    DEC  RF    ;Transfer
  65   30    BN         ;Loop until transfer complete
  66   55
  67   D4    SEP  R4    ;Begin exit
  68   01               ;Call Erase to End of Line (deleted line)
  69   77
  6A   30    BN         ;Branch to 0121 to display memory & return
  6B   21                 (This branch conserves memory)
                       DELETE AND CLOSE UP LINE

016C   D4    SEP  R4
  6D   00               ;Call Close Up Routine @ 0097  (no more room
  6E   97               ;Exists on this page for the sub
  6F   F8    LDI
0170   09               ;(09-4K/05-3K)
  71   BA    PHI  RA    ;Set RA = last line in Text Buffer
  72   F8    LDI        ;    "          "        "     "
  73   F0               ;    "          "        "     "
  74   AA    PLO  RA    ;    "          "        "     "
  75   30    BN         ;Branch to 0167 to exit and to
  76   67               ;Call Erase to End of Line (If this were not
                          done, the last line would be repeated)

                        ERASE TO END OF LINE

0177   E2    SEX  2     ;X = 2
  78   87    GLO  R7
  79   73    STXD       ;Push R7.0
  7A   8A    GLO  RA
  7B   73    STXD       ;Push RA.0
  7C   F8    LDI
  7D   20               ;Load R8.1 with ASCII space code
  7E   B8    PHI  R8
  7F   D4    SEP  R4
0180   03               ;Call Display New Character at Cursor
  81   9F
  82   8A    GLO  RA
  83   FA    ANI        ;Test if RA.0 = NF  (at line end)
  84   0F               ;First stripping the first digit
  85   FB    XRI        ;Test
```

```
0186   0F
  87   32    BZ          ;If so, branch to reset cursor & RA - done
  88   91
  89   1A    INC   RA    ;Data pointer + 01 - next character
  8A   8A    GLO   RA
  8B   F6    SHR         ;Test if RA is even
  8C   33    BDF
  8D   8F                ;If not branch (character in right - do not INC R7)
  8E   17    INC   R7
  8F   30    BN          ;Loop till line is erased
0190   7C
  91   E2    SEX   2     ;X = 2
  92   60    IRX         ;Point to saved data
  93   72    LDXA        ;Pop RA.0
  94   AA    PLO   RA    ;Restore data pointer
  95   F0    LDX         ;Pop R7.0
  96   A7    PLO   R7    ;Restore cursor address
  97   D5    SEP   R5    ;Return

                          ERASE TO END OF PAGE

0198   E2    SEX   2     ;X = 2
  99   97    GHI   R7
  9A   73    STXD        ;Save R7 on stack  (old cursor)
  9B   87    GLO   R7    ;Push R7.1
  9C   73    STXD        ;Push R7.0
  9D   9A    GHI   RA    ;Push RA.1
  9E   73    STXD        ;Save RA on stack  (old data pointer)
  9F   8A    GLO   RA    ;Push RA.0
01A0   73    STXD
  A1   89    GLO   R9
  A2   FC    ADI
  A3   F0                ;Add F0 (240 = 16 character x 15 lines)
  A4   AC    PLO   RC    ;To R9 = last text row of any page
  A5   99    GHI   R9
  A6   7C    ADCI
  A7   00
  A8   BC    PHI   RC    ;RC points to last row of text (RC only used in key
  A9   D4    SEP   R4                                    scan routines free here)
  AA   01                ;Call Delete to End of Line
  AB   77
  AC   E2    SEX   2     ;X = 2
  AD   9A    GHI   RA
  AE   52    STR   R2    ;Push RA.1 to test
  AF   9C    GHI   RC
01B0   F3    XOR         ;If = RC.1 (last line of text pointer)
  B1   3A    BNZ         ;If not =, continue
  B2   B9
  B3   8A    GLO   RA
```

```
01B4   52    STR   R2    ;Push RA.0 to test
  B5   8C    GLO   RC
  B6   F3    XOR         ;If = RC.0 (last line of text pointer)
  B7   32    BZ          ;If = (RA=RC) then done - branch to exit
  B8   BE                ;Else continue
  B9   D4    SEP   R4
  BA   02                ;Call Carriage Return for next line
  BB   98
  BC   30    BN          ;Loop till done
  BD   A9
  BE   D4    SEP   R4
  BF   03                ;Call Display Cursor to erase
01C0   5C
  C1   E2    SEX   2     ;X =2
  C2   60    IRX         ;Point to saved data
  C3   72    LDXA        ;Pop RA.0
  C4   AA    PLO   RA
  C5   72    LDXA        ;Pop RA.1
  C6   BA    PHI   RA    ;Restore RA
  C7   72    LDXA        ;Pop R7.0
  C8   A7    PLO   R7
  C9   F0    LDX         ;Pop R7.1
  CA   B7    PHI   R7    ;Restore R7
  CB   D4    SEP   R4
  CC   03                ;Call Display Cursor
  CD   5C
  CE   D5    SEP   R5    ;Return

                           ERASE TEXT BUFFER

01CF   F8    LDI
  D0   04
  D1   BE    PHI   RE    ;Set RE, R9 = 0400 = beginning Text Buffer
  D2   B9    PHI   R9    ;     "       "          "          "
  D3   F8    LDI         ;     "       "          "          "
  D4   00                ;     "       "          "          "
  D5   AE    PLO   RE    ;     "       "          "          "
  D6   A9    PLO   R9    ;     "       "          "          "
  D7   F8    LDI
  D8   20                ;ASCII code for space
  D9   5E    STR   RE    ;Store via RE in text
  DA   1E    INC   RE    ;Re + 1  next character position
  DB   9E    GHI   RE
  DC   FB    XRI         ;Test if RE went past last page of text
  DD   06/0A
  DE   3A    BNZ         ;If not, loop until all text set to ASCII 20's
  DF   D7
01E0   30    BN          ;Branch to exit point, displaying memory page,
  E1   21                ;Homing cursor, and return @ 0121
```

```
                        CALL TO TAPE READ/WRITE

01E2    D4    SEP   R4
  E3    00                ;Call Tape Read/Write @ 0050
  E4    50                ;(Because no more room exists on this page)
  E5    D5    SEP   R5    ;Return

                        CALL TO CURSOR ON/OFF

01E6    D4    SEP   R4
  E7    03                ;Call Display Cursor to erase at present
  E8    5C                ;Position
  E9    D4    SEP   R4
  EA    03                ;Call Cursor On/Off to switch
  EB    C0
  EC    D4    SEP   R4
  ED    03                ;Call Home Cursor
  EE    53
  EF    D5    SEP   R5    ;Return

                        CALL TO SHOW PAGE "N"

01F0    D4    SEP   R4
  F1    03                ;Call Show Page "N"  (needed as there is
  F2    D3                ;No more room on this page)
  F3    30    BN          ;Branch to set R9.0 = 00, display memory,
  F4    32                ;Home cursor and return @ 0132

                             HOME CURSOR

01F5    D4    SEP   R4
  F6    03                ;Call Display Cursor to erase
  F7    5C
  F8    30    BN          ;Branch to Home Cursor  (in Cursor On/Off routine)
  F9    EC
```

01FA-FF Available for calls to two more function routines-see instructions for using this expansion area

```
                     FUNCTION DECODE - FOR KEYS 8-F

0200    93    GHI   R3    ;(02) (PC)
  01    BE    PHI   RE    ;RE.1 = 02
  02    98    GHI   R8    ;(Key press)
  03    AE    PLO   RE    ;RE points to look up table for function address
  04    0E    LDN   RE    ;Get function address
  05    A3    PLO   R3    ;Put in R3.0 - jump to function routine
  06    00  Filler
  07    D5    SEP   R5    ;Effects return for disabled functions (set any ML
                                                          0208-020F = 07)
```

```
0208   DE   Cursor Left
  09   C6   Cursor Right
  0A   10   Scroll Forward          Function address look up table-
  0B   44   Scroll backward         Disabled functions set = 07 for
  0C   F0   Control (C)             return via D5 @ ML 0207
  0D   98   Carriage Return
  0E   7F   Cursor Up
  0F   5F   Cursor Down

            SCROLL FORWARD - CURSOR POSITION UNCHANGED
                      (Error tone @ 023B)

0210   E2   SEX  2      ;X = 2
  11   99   GHI  R9
  12   FB   XRI         ;Test if R9.1 points
  13   05/09            ;To last text page (3K=05/4K=09)
  14   32   BZ          ;If so, branch to exit & sound tone
  15   3B
  16   97   GHI  R7
  17   73   STXD        ;Push R7.1
  18   87   GLO  R7
  19   73   STXD        ;Push R7.0  (save old cursor address)
  1A   9A   GHI  RA
  1B   73   STXD        ;Push RA.1
  1C   8A   GLO  RA
  1D   73   STXD        ;Push RA.0  (save old data pointer)
  1E   89   GLO  R9
  1F   FC   ADI
0220   10               ;Add 10 hex (16 decimal)
  21   A9   PLO  R9
  22   99   GHI  R9     ;To R9 - to point
  23   7C   ADCI
  24   00               ;To new text page beginning
  25   B9   PHI  R9
  26   D4   SEP  R4
  27   03               ;Call Display Memory - one text page (scrolled up)
  28   2F
  29   E2   SEX  2      ;X = 2
  2A   60   IRX         ;Point to data
  2B   72   LDXA        ;Pop RA.0
  2C   FC   ADI
  2D   10
  2E   AA   PLO  RA
  2F   72   LDXA        ;Pop RA.1
0230   7C   ADCI
  31   00
  32   BA   PHI  RA     ;Restore data pointer-(add 10 hex for the scroll
  33   72   LDXA        ;Pop R7.0                              operation)
  34   A7   PLO  R7
  35   F0   LDX         ;Pop R7.1  (last pop-stack clear)
```

```
0236   B7   PHI  R7   ;Restore old cursor address
  37   D4   SEP  R4
  38   03              ;Call Display Cursor
  39   5C
  3A   D5   SEP  R5   ;Return
  3B   7B   SEQ       ;Tone on for end of text warning
  3C   F8   LDI
  3D   3C              ;Value for 1 second tone
  3E   A8   PLO  R8   ;Put into timer
  3F   88   GLO  R8
0240   3A   BNZ       ;Loop until timer = 00
  41   3F
  42   7A   REQ       ;Tone off
  43   D5   SEP  R5   ;Return

                  SCROLL BACKWARD - CURSOR HOMED

0244   89   GLO  R9
  45   FF   SMI
  46   10              ;Subtract 16 decimal
  47   A9   PLO  R9
  48   99   GHI  R9   ;From R9 (data page pointer)
  49   7F   SMBI
  4A   00              ;To point back one row
  4B   B9   PHI  R9
  4C   FB   XRI
  4D   03              ;Test if R9 went past first page
  4E   32   BZ
  4F   57              ;Boundary (0400)-If so, branch to error & return
0250   D4   SEP  R4                          (resetting R9 = 0400)
  51   03              ;Call Display Memory - one text page (scrolled back)
  52   2F                                     (cursor off)
  53   D4   SEP  R4
  54   03              ;Call Home Cursor      (cursor on)
  55   53
  56   D5   SEP  R5   ;Return
  57   F8   LDI
  58   04              ;Reset R9
  59   B9   PHI  R9
  5A   F8   LDI       ;To first text page
  5B   00
  5C   A9   PLO  R9   ;At 0400
  5D   30   BN
  5E   3B              ;Branch to error tone @ 023B & return

                       CURSOR DOWN

025F   D4   SEP  R4
  60   03              ;Call Display Cursor to erase old
```

```
0261   5C
  62   87   GLO  R7    ;Entry ROM carriage return-(cursor already off)
  63   FC   ADI
  64   40             ;Add 64 decimal to
  65   A7   PLO  R7
  66   97   GHI  R7    ;Cursor to point
  67   7C   ADCI
  68   00             ;To next row
  69   B7   PHI  R7
  6A   FB   XRI        ;Test if Display Cursor about to go off page
  6B   0C/10           ;At bottom (change to 10 hex for 4K)
  6C   32   BZ
  6D   7A              ;If so, branch to error exit, homing cursor
  6E   8A   GLO  RA
  6F   FC   ADI        ;Else add
0270   10
  71   AA   PLO  RA    ;16 decimal to RA data pointer
  72   9A   GHI  RA
  73   7C   ADCI       ;For next row
  74   00
  75   BA   PHI  RA    ;Of text in memory
  76   D4   SEP  R4
  77   03             ;Call Display Cursor
  78   5C
  79   D5   SEP  R5    ;Return
  7A   D4   SEP  R4    ;Error exit for  ;cursor up/cursor down
  7B   03              ;Call Home Cursor
  7C   53
  7D   30   BN         ;Branch to error tone
  7E   3B              ;And return

                              CURSOR UP

027F   D4   SEP  R4
  80   03             ;Call Display Cursor to erase old
  81   5C
  82   87   GLO  R7
  83   FF   SMI        ;Subtract 64 decimal
  84   40
  85   A7   PLO  R7    ;From cursor
  86   97   GHI  R7
  87   7F   SMBI       ;To point to next row up
  88   00
  89   B7   PHI  R7    ;Test if R7.1 went past
  8A   FB   XRI
  8B   07/0B           ;Upper page boundary (3K-07/4K-0B)
  8C   32   BZ         ;If so, branch to
  8D   7A              ;Error message rehoming cursor (in Cursor Down sub)
  8E   8A   GLO  RA
  8F   FF   SMI        ;Else Subtract 16 decimal
```

```
0290   10
  91   AA    PLO  RA    ;From RA data pointer
  92   9A    GHI  RA
  93   7F    SMBI       ;To point to next row of
  94   00
  95   BA    PHI  RA    ;Text in memory (up)
  96   30    BN         ;Branch to
  97   76               ;Display Cursor & return (in Cursor Down sub)

                            CARRIAGE RETURN

0298   D4    SEP  R4
  99   03               ;Call Display Cursor to erase old
  9A   5C
  9B   87    GLO  R7    ;Enter here if cursor already off
  9C   FA    ANI        ;Mask last 3 bits R7.0
  9D   07
  9E   32    BZ         ;Branch if = 00 -- when last three bits R7 = 00
  9F   A3               ;Then cursor is at extreme left of display
02A0   27    DEC  R7    ;R7 - 01 to move left
  A1   30    BN
  A2   9B               ;Loop till done
  A3   8A    GLO  RA
  A4   FA    ANI        ;Mask last 4 bits RA.0
  A5   0F
  A6   32    BZ         ;Branch if = 00 -- when last 4 bits RA = 00
  A7   AB               ;Then data pointer is at the beginning of a row
  A8   2A    DEC  RA    ;RA -01  Data pointer left (of row)
  A9   30    BN
  AA   A3               ;Loop till done
  AB   87    GLO  R7
  AC   FC    ADI
  AD   40               ;Add 40 hex  64 decimal to value in R7 cursor
  AE   97    GHI  R7
  AF   7C    ADCI       ;To test if at page bottom - R7 unchanged
02B0   00               ;And (cursor off) still
  B1   FB    XRI        ;When R7.1 would = 0C, then cursor is at
  B2   0C/10             ;(3K=0C/4K=10)Page bottom, and a scroll up must be
  B3   3A    BNZ        ;If ≠ 00 (R7.1 would ≠ 0C) branch to      done
  B4   BE               ;Simply move cursor down a row
  B5   99    GHI  R9    ;Test if on
  B6   FB    XRI
  B7   05/09             ;Last text page (3K=05/4K=09)
  B8   32    BZ
  B9   C2               ;Branch to turn on cursor-no scroll-no cursor down
  BA   D4    SEP  R4
  BB   02               ;Call Scroll Up - auto with carriage return
  BC   10               ;At page bottom
  BD   D5    SEP  R5    ;Return
```

```
02BE   D4   SEP  R4
  BF   02             ;Call Cursor Down (skipping the turn off cursor
02C0   62             ;Normal mid page carriage returns        command)
  C1   D5   SEP  R5   ;Return
  C2   D4   SEP  R4
  C3   03             ;Call Display Cursor
  C4   5C             ;When at page bottom last page text
  C5   D5   SEP  R5   ;Return

                           CURSOR RIGHT

02C6   D4   SEP  R4
  C7   03             ;Call Display Cursor to erase old
  C8   5C
  C9   8A   GLO  RA   ;Test if RA is even/odd
  CA   F6   SHR
  CB   3B   BNF       ;If even (DF=0), branch
  CC   D9             ;To increment data pointer (cursor moves within byte)
  CD   8A   GLO  RA
  CE   FA   ANI       ;Else test if at the extreme right
  CF   0F
02D0   FB   XRI       ;(RA.0 = NF at the end of rows)
  D1   0F
  D2   3A   BNZ       ;If not at row end, skip Carriage Return command
  D3   D8
  D4   D4   SEP  R4
  D5   02             ;Call Carriage Return with auto scroll at page end
  D6   9B             ;(Skipping the turn off cursor command)
  D7   D5   SEP  R5   ;Return (cursor on - all pointers correct)
  D8   17   INC  R7   ;Display Cursor + 01 (next byte over)
  D9   1A   INC  RA   ;Data pointer + 01 (next in row)
  DA   D4   SEP  R4
  DB   03             ;Call Display Cursor
  DC   5C
  DD   D5   SEP  R5

                           CURSOR LEFT

02DE   D4   SEP  R4
  DF   03             ;Call Display Cursor to erase old
02E0   5C
  E1   8A   GLO  RA
  E2   F6   SHR       ;Test RA for even/odd
  E3   33   BDF       ;Branch if odd (cursor in right of byte)
  E4   EB
  E5   8A   GLO  RA
  E6   FA   ANI       ;Mask last 4 bits RA
  E7   0F
  E8   32   BZ        ;If = 00, then cursor is at extreme left
```

```
02E9    EC                  ;And no operation is performed
  EA    27    DEC   R7      ;Display Cursor - 01 (one byte left)
  EB    2A    DEC   RA      ;Data pointer - 01   (one character left)
  EC    D4    SEP   R4
  ED    03                  ;Call Display Cursor @ New Position
  EE    5C
  EF    D5    SEP   R5      ;Return
```

CONTROL (C) SELECT

```
02F0    D4    SEP   R4
  F1    02                  ;Call Error tome - warn user that a
  F2    3B                  ;Control function will be selected
  F3    D4    SEP   R4
  F4    01                  ;Call Control (C) Function Decode and
  F5    10                  ;Perform the function
  F6    D5    SEP   R5      ;Return

02F7-02FF  Available
```

4-PAGE INTERRUPT WITH TIMER

```
0300    72    LDXA          ;Restore D  ;Exit
  01    70    RET           ;Return
  02    C4    NOP           ;Entry point
  03    22    DEC   R2      ;Stack free
  04    78    SAV           ;Push T
  05    22    DEC   R2
  06    52    STR   R2      ;Push D
  07    88    GLO   R8      ;R8 is decrementing timer
  08    32    BZ            ;Auto - 01 until = 00
  09    0B
  0A    28    DEC   R8      ;R8 - 01
  0B    9B    GHI   RB      ;RB→R0 (DMA prepare)
  0C    B0    PHI   R0      ;  "      "     "
  0D    F8    LDI           ;  "      "     "
  0E    00                  ;  "      "     "
  0F    A0    PLO   R0      ;  "      "     "
0310    30    BN            ;Branch
  11    00                  ;To exit
```

CALL ROUTINE - RUNS IN R4

```
0312    D3    SEP   R3      ;Exit/Call sub routine with R3 = PC
  13    E2    SEX   R2      ;X = 2  ;Entry point
  14    96    GHI   R6
  15    73    STXD          ;Push R6.1
  16    86    GLO   R6
  17    73    STXD          ;Push R6.0
```

```
0318   93   GHI  R3   ;R3→R6
 19    B6   PHI  R6   ;Save the return address
 1A    83   GLO  R3
 1B    A6   PLO  R6
 1C    46   LDA  R6   ;Load sub routine
 1D    B3   PHI  R3   ;Address
 1E    46   LDA  R6   ;Into R3
 1F    A3   PLO  R3   ;  "   "
0320   30   BN        ;Branch to exit
 21    12

                  RETURN ROUTINE - RUNS IN R5

0322   D3   SEP  R3   ;Exit/to "Main" (calling) routine
 23    96   GHI  R6   ;R6→R3  ;Entry point
 24    B3   PHI  R3   ;R3 = Return address
 25    86   GLO  R6   ;  "      "        "
 26    A3   PLO  R3   ;  "      "        "
 27    E2   SEX  R2   ;X = 2
 28    60   IRX       ;Point to saved R6
 29    72   LDXA      ;Pop R6.0
 2A    A6   PLO  R6   ;Restore R6.0
 2B    F0   LDX       ;Pop R6.1
 2C    B6   PHI  R6   ;Restore R6.1
 2D    30   BN        ;Branch to exit
 2E    22

DISPLAY MEMORY PAGE - ONE PAGE DISPLAYED-CURSOR OFF ON RETURN

032F   D4   SEP  R4
 30    03             ;Call Home Cursor (no need to erase old cursor)
 31    53
 32    D4   SEP  R4
 33    03             ;Call Erase Display
 34    AE             ;(Erases all old cursors, of course, too)
 35    0A   LDN  RA   ;Get a character
 36    B8   PHI  R8   ;R8.1 is ASCII holder for display
 37    D4   SEP  R4
 38    03             ;Call Display Character @ R7
 39    5F
 3A    1A   INC  RA   ;Next character
 3B    8A   GLO  RA
 3C    F6   SHR       ;Test LSB of RA.0
 3D    33   BDF       ;If odd, (DF=01) then loop is only on first swing
 3E    35             ;And a second character will go in R7 on display
 3F    17   INC  R7   ;Display Cursor + 01
0340   87   GLO  R7
 41    FA   ANI
 42    07             ;Mask all but last 3 bits to test if at begin of line
```

```
0343   3A    BNZ          ;If not, then line is not finished
  44   35                 ;Continue
  45   87    GLO   R7
  46   FC    ADI          ;Else add 38 hex (56 decimal)
  47   38
  48   A7    PLO   R7     ;To cursor
  49   97    GHI   R7
  4A   7C    ADCI         ;To point to next
  4B   00
  4C   B7    PHI   R7     ;Row
  4D   97    GHI   R7
  4E   FB    XRI          ;Then test if R7.1 went past last page bottom
  4F   0C/10               ;(3K=0C/4K=10)
0350   3A    BNZ          ;Continue to loop
  51   35                 ;Till page is displayed
  52   D5    SEP          ;Return

            HOME CURSOR-CURSOR MUST BE OFF ON ENTRY

0353   9B    GHI   RB
  54   B7    PHI   R7     ;R7.1 = RB.1
  55   F8    LDI
  56   00
  57   A7    PLO   R7     ;R7 = RB  -Homes Display cursor
  58   99    GHI   R9
  59   BA    PHI   RA
  5A   89    GLO   R9
  5B   AA    PLO   RA     ;RA = R9  -Top byte in data page - homes data
                                      displacement pointer
                                    (continue on to Display Cursor routine)

          DISPLAY CURSOR (ALSO ERASES CURSOR IF RECALLED)

035C   F8    LDI
  5D   5F                 ;Load ASCII underline (change to 20 for invisible)
  5E   B8    PHI   R8     ;Set into R8.1 for display (or erasing)
                                    (continue on to Display Character sub)

        DISPLAY CHARACTER - DISPLAYS ASCII CHARACTER IN R8.1

035F   98    GHI   R8     ;R8.1 holds ASCII code
  60   FE    SHL          ;Multiply x 4 (4 bytes to each bit pattern)
  61   FE    SHL
  62   AF    PLO   RF     ;Put in RF.0 index
  63   F8    LDI          ;Load page address of character set into RF.1
  64   06/0A                ;(3K=06/4K=0A)
  65   7C    ADCI         ;Adding possible carry from the multiply
  66   00
  67   BF    PHI   RF     ;RF now indexes the character bit pattern
```

```
0368   F8   LDI
 69    04
 6A    AE   PLO  RE    ;Set loop counter = 04
 6B    0F   LDN  RF    ;Get character bit pattern (2 rows)
 6C    FA   ANI       ;AND with F0 for MSB's
 6D    F0
 6E    BE   PHI  RE    ;Store in RE.1 to pass to sub
 6F    D4   SEP  R4
0370   03             ;Call Display Bit Row @ R7
 71    88
 72    4F   LDA  RF    ;Get same bit pattern
 73    FE   SHL       ;Shift left for LSB's
 74    FE   SHL       ;  "     "       "
 75    FE   SHL       ;  "     "       "
 76    FE   SHL       ;  "     "       "
 77    BE   PHI  RE    ;Store in RE.1 to pass to sub
 78    D4   SEP  R4
 79    03             ;Call Display Bit Row @ R7
 7A    88
 7B    2E   DEC  RE    ;Loop count - 01
 7C    8E   GLO  RE
 7D    3A   BNZ       ;Loop 4 times total to display eight bit rows
 7E    6B
 7F    87   GLO  R7
0380   FF   SMI       ;Subtract 40 hex from cursor address
 81    40
 82    A7   PLO  R7    ;To reset to top of character rows
 83    97   GHI  R7
 84    7F   SMBI      ;(This may be shortened by 3 bytes by using
 85    00             ;A separate register for display)
 86    B7   PHI  R7
 87    D5   SEP  R5    ;Return

  DISPLAY BIT ROW-DOES ONE OF 8 BIT ROW PATTERNS FOR A CHARACTER

0388   8A   GLO  RA    ;Data displacement pointer
 89    F6   SHR       ;Shift to test even/odd
 8A    3B   BNF       ;Branch if even (DF=0)
 8B    92             ;To skip the shifting (display bits on left)
 8C    9E   GHI  RE    ;Get unpacked bits in RE.1 (RA is odd)
 8D    F6   SHR       ;Shift right x 4 (display bits on right)
 8E    F6   SHR
 8F    F6   SHR
0390   F6   SHR
 91    BE   PHI  RE    ;Store processed bit row in RE.1
 92    E7   SEX  7     ;X = 7  (R7 is display cursor)
 93    9E   GHI  RE    ;Get the processed bit row
 94    F3   XOR       ;Exclusive OR with what is already displayed
 95    57   STR  R7    ;Put in display @ R7
```

```
0396  87   GLO  R7
  97  FC   ADI       ;Add 08 to cursor address to
  98  08
  99  A7   PLO  R7   ;Point to next bit row for characters
  9A  97   GHI  R7
  9B  7C   ADCI
  9C  00             ;Adding possible carry to R7.1
  9D  B7   PHI  R7
  9E  D5   SEP  R5   ;Return

      NEW CHARACTER AT CURSOR-FOR ENTERING NEW CHARACTERS

039F  98   GHI  R8   ;Get new ASCII
  A0  52   STR  R2   ;Push-(for immediate use)
  A1  0A   LDN  RA   ;Get old ASCII in text
  A2  B8   PHI  R8   ;Put in R8.1
  A3  02   LDN  R2   ;Pop new ASCII
  A4  5A   STR  RA   ;Store in text at memory displacement pointer
  A5  D4   SEP  R4
  A6  03             ;Call Display to Erase old character @ cursor
  A7  5F
  A8  0A   LDN  RA   ;Get new ASCII from text
  A9  B8   PHI  R8   ;Put in R8.1
  AA  D4   SEP  R4
  AB  03             ;Call Display to Show New character @ cursor
  AC  5F
  AD  D5   SEP  R5   ;Return

          ERASE DISPLAY PAGES-SETS 4 REFRESH PAGES = 00

03AE  9B   GHI  RB
  AF  BF   PHI  RF   ;RF.1 = RB.1 -top display page
03B0  FC   ADI
  B1  04
  B2  E2   SEX  2    ;Add 4 to this value to indicate when
  B3  52   STR  R2   ;Pointer goes past last page-push on stack
  B4  F8   LDI
  B5  00
  B6  AF   PLO  RF   ;RF now initialized to first display byte
  B7  F8   LDI
  B8  00             ;Get 00 for erasing
  B9  5F   STR  RF   ;Store in display area
  BA  1F   INC  RF   ;RF + 01
  BB  9F   GHI  RF
  BC  F3   XOR       ;Test if RF.1=stacked byte and is off bottom
  BD  3A   BNZ       ;If not, loop until done
  BE  B7
  BF  D5   SEP  R5   ;Else return
```

CURSOR ON/OFF-COMPLIMENTS CURSOR FOR INVISIBLE OR UNDERLINE DISPLAY

```
03C0   F8   LDI
  C1   03
  C2   BF   PHI  RF   ;RF points to memory byte @ 035D
  C3   F8   LDI
  C4   5D
  C5   AF   PLO  RF   ;Containing ASCII code for cursor
  C6   0F   LDN  RF   ;Get the current cursor code
  C7   FB   XRI       ;Exclusive OR with "20" (ASCII space)
  C8   20             ;To test if cursor is on or off
  C9   32   BZ        ;If = 00, cursor is off - branch to turn on cursor
  CA   CF
  CB   F8   LDI       ;If ≠ 00 cursor is on, load "20" ASCII space
  CC   20
  CD   30   BN
  CE   D1
  CF   F8   LDI       ;If cursor is off, load 5F - ASCII underline
03D0   5F
  D1   5F   STR  RF   ;Store D via RF - cursor either on or off
  D2   D5   SEP  R5   ;Return

            SHOW PAGE "N"-ENTER # PAGE FOR DISPLAY
                        (1-6 4K  1-2  3K)
03D3   DC   SEP  RC   ;Do key scan routine in ROM for page #
  D4   32   BZ        ;If "0" page selected, branch to exit ignoring
  D5   DF             ;The bad command
  D6   F0   LDX       ;Get page number (stacked by ROM routine)
  D7   FD   SDI       ;Subtract: 6-number (if negative, number>6)
  D8 02/06            ;(3K=02/4K=06)
  D9   3B   BM        ;If negative, branch to exit ignoring
  DA   DF             ;The bad command
  DB   F0   LDX       ;Once again get the page number (which is = 1-6)
  DC   FC   ADI       ;Add 3 to reference the correct memory page
  DD   03             ;(1=04; 2=05; 3=06; etc.)
  DE   B9   PHI  R9   ;R9 points to correct page top
  DF   D5   SEP  R5   ;Return
```

03E0-03FF 32 bytes available for expansion

D I S A S S E M B L E R 7

D I S A S S E M B L E R - 7

INTRODUCTION

One day, right in the midst of moving a block of Chip-8 data around with an editor I had written, a dirty little bug surfaced chewing holes in my data with an appetite held at bay only by my lack of more than 4K of RAM to eat. I had written the editor months earlier -- and never had any problems with it.

But I keep good notes on all my programs, and as the experts will tell you to do, lace my instructions with more comments than a friend of mine laces his coffee with brandy. (A good analogy only if you could see my friend slosh brandy into his coffee.)

Never have I heard any expert tell how to avoid losing your notes, however. Laced or unlaced they don't do much good in some drawer in Pennsylvania when you happen to be living in the Sierra Madres in Mexico.

It took me an hour and a half to write out the program steps to a program occupying a little over a page and a half of memory. And the bug was so simple, once I found it, that if I had had a disassembled listing of the program, I'm sure I could have uncovered the dark cave of its hide-a-way in a few minutes.

So I wrote the DISASSEMBLER-7, which takes its name (like the TEXT EDITOR 21) from the number of functions available for use.

Operation is straightforward. All of RCA's standard mnemonics have been used. Up to 3 pages of machine language program can be disassembled at once, with the ability to show from 16 to 48 bytes and addresses on the screen at one time. That's up to 48 pages of displayed data!

In addition to simply viewing data, you have the option to enter bytes making the disassembler useful as a debugging tool. Each function is displayed on screen as it is selected, so you always know what you're about to do - before you do it.

With the addition of tape I/O routines, DISASSEMBLER-7 makes it a breeze to take programs apart. If you're an old pro at machine language programming, you'll appreciate the versatility of this program. If you're just learning, you'll find it an invaluable tool to your progress. Good luck with your programming!

OPERATION

Written for 4K Cosmac VIP systems, DISASSEMBLER-7 takes any machine language program and converts it to standard 1802 mnemonics for viewing and editing. At any time, you may specify the starting address of your program so if your program was written to begin at 0500, you don't have to pretend that 0000 means 0500. You can display from any address and the program tells you when you have selected an address that is out of range. For programs longer than 3 pages, the tape read and write

routines allow chaining sections of programs together for fast disassembly. If you have a printer, a simple routine could be added to output the contents of the text buffer page to add a professional touch to your programs.

In addition to the DISASSEMBLER-7 program, you need to enter a 2-page character generator and a 2-page mnemonic look-up table at the locations specified in the typed listing. These have been provided with the program on tape, but are not listed here. (See program description for formatting the look up table.) The table forms the heart of DISASSEMBLER-7, and is completely user definable allowing you to use your own mnemonics if you prefer, and to specify which of <u>any</u> bytes you want to be followed by one or two bytes of data.

(The following detailed description of DISASSEMBLER-7's operation should be studied carefully in order to fully appreciate the program's capability.)

If you have a copy of the tape, you will also have a sample listing containing 3 pages of data to be entered. Flip the run switch up and the words "READ/WRITE?" should appear. (If you have difficulties, please refer to the checksum data at the end of the program description.)

Set your tape recorder to the beginning of the sample or to any machine language program that is at <u>least</u> 3 pages in length. (The beginning of any Chip-8 game will do, and the first issue of VIPER may be used to follow the disassembled program flow of the Chip-8

Interpreter) Press Key B and the screen will clear. Start the tape. When the program has been entered, control is automatically given back to DISASSEMBLER-7, and your program will appear on the screen. (Don't forget to stop the tape.)

At this point you have several options:

1) Page forward -- Pressing Key C will cause the program to be displayed from the address immediately following the previously displayed last address on screen. Paging past the third page of your program will cause a wrap-around to the beginning.

2) Enter Show From Address -- Press Key D, then the address you wish to show from. The selected function ("SHOW FROM") and the address are displayed on the screen. If you enter an address out of range of the 3 pages in your program, an "ERROR-RETRY" message is displayed and the program is redisplayed. (Try entering FFFF to examine this feature.)

If when viewing the very last page of your program, you enter a new show from address that is not in range, the program will start from a faulty location displaying only a single address on screen (0300 if your program began at 0000.) This can easily be rectified by simply entering a correct show from address (Key D, then the address.)

3) Enter Start From Address -- this address, which will be referred to as the PSEUDO ADDRESS in the program listing, allows you to tell the computer the address that your machine language program is written from. In other words, if you have a program that starts at 0500 such as the Register Display and Breakpoint program in the NOV. 78 VIPER, you wouldn't want it to be disassembled at 0000. Therefore after loading the program (using DISASSEMBLER's Tape Read feature) you need to tell the computer to start from 0500 by pressing Key E, then the address.

All Start From addresses must be from a page beginning -- 0200, 0300 etc. Entering 0252 for instance will read 0200 to conform with the VIP's recording scheme. (This may be disabled. See program description.) Remember, this is only a Pseudo Address and does not represent an actual place in memory.

4) Write Byte -- pressing Key 0 calls this editing routine which uses the Pseudo Address to enter bytes in your program. Enter an address, then the byte you wish to enter. If you change your mind press any key but E. Key E will write the byte at the specified Pseudo Address -- in other words, at the correct place in your program. (Where else?)

I stress this because you may think since your data is actually in 0500-07FF, changing the byte at 0500 will alter the first byte. But since you tell the computer where to begin disassembly, if you tell it to begin at 0700 for instance, then, that is the first address of your program. The best way to use the DISASSEMBLER is to forget about it being there. Think only of your program and DISASSEMBLER-7 will do the rest.

If you enter an address that is out of range when selecting WRITE BYTE, the program will reset to display the listing. This allows an easy escape if you change your mind as well. Also, upon pressing Key E to enter the byte, the program will display the disassembled flow from that byte.

5-6) TAPE READ/WRITE --This functions much in the same way as for TEXT EDITOR-21. The difference is that all tape operations are fixed at 3 pages instead of at 6 as for the EDITOR. You may be wondering, then, how to view programs that are less than 3 pages long. One way, the best, is to record 3 pages on tape including your shorter program, then read those pages in using DISASSEMBLER-7's tape read. Another would be to "hand load" the program using the ROM monitor, but be sure to load only into ML's 0500-07FF. Still another would be

to simply run the shorter program in which will cause an error tone to kick on. No problem. Simply reset the computer and the shorter program will all be there.

On initial power up of DISASSEMBLER-7, Tape Read/Write is called as previously mentioned. Hoever, if at any time you wish to cancel the tape operation, pressing any key other than Keys B or F will restart the program listing, escaping the Tape Select feature.

When writing to tape, 3 pages will be output starting as soon as you hit Key F so be sure to start the tape recorder before pressing the key. If you output a listing to tape while viewing from an address not on the first page of your listing, the program will be listed from an incorrect Pseudo Address after the taping operation. It is best, therefore to always set the listing (using the SHOW FROM Function) to the beginning address when outputting to tape. Of course START FROM may be used to correct an improper pseudo address at any time, and the actual output to tape is never affected in any case.

When reading from tape, you need to consider one of two things. Is the program to be disassembled self-contained, or is it only part of a longer program? If it is separate and will not be followed by additinnal data, then simply read it in and enter the correct START FROM or pseudo address when done with the tape read.

(on power up, this is automatically 0000)

If, however, the listing is part of a longer program, first enter the beginning 3 pages. Before reading in the next portion, be sure that the last address (02FF when starting from 0000) is displayed on the screen. It does not have to be in any special place, just be sure it is there. After reading in the next portion of you program, the correct pseudo address will be displayed and you can continue to view the listing. (Following the example, the first section ended at 02FF, so the next will start at 0300). Again, no matter what you do here, the program will be read in correctly, provided of course that your little brother wasn't playing with his magnetic doggies on top of your cassette cases again. If the start address is incorrect, simply adjust it using START FROM.

7) Add Table Entry -- I've not discussed the display format of DISASSEMBLER-7 until here for a good reason. If you have been viewing programs before reading this far, you will notice that some instructions are displayed alone while others are followed by one or two bytes such as this: 0000 F8FF LDI ,or, 0200 C0 0406 LBR. This is because some instructions require the next one or two bytes for operation. In the first example, F8 or its mnemonic LDI means "load the D register with the following data." The following data is FF, and that is

all it is -- data. In the second example, the C0 is a long branch instruction requiring 2 bytes of data. This data, which immediately follows the C0 instruction is an address - 0406 - and the program would jump to this address. If DISASSEMBLER-7 did not recognize this fact, it could possibly view data as instructions causing a listing which would be hopelessly confusing to read.

Therefore, DISASSEMBLER-7 contains a feature which makes it impossible to view from the middle of data. To illustrate what I mean, read in the first three pages of DISASSEMBLER-7 for disassembly, and follow along in the typed listing.

Notice that at ML 0009, we have the instruction F8FF which, as previously explained, means load the D register with the byte FF. Now press Key D to select SHOW FROM. Enter the address 000A and you will see that the program recognizes the byte at 000A to be data -- it is not an instruction -- so it displays from 0009 even though we requested 000A. In the event that two such instructions occur back to back, the listing will <u>always</u> begin with the first instruction it can be sure is not a piece of data to another instruction. (These data bytes, by the way, are commonly referred to as arguments.) Try -- but not just now -- to load a page with C0 C0 C0, long branches to the address C0 C0, then attempt to view from the middle

of the page. The computer keeps backing up to be sure it is always displaying a valid instruction.

Now press Key C Page Forward to display from ML 001F. At ML 0025 you will see a D4, SEP; 03 LDN; and a 3069 BR, branch to ML 0069. Now check the typed program listing to confirm this. What? You say this is not correct? Horrors! What went wrong?

Both DISASSEMBLER-7 and TEXT EDITOR-21 use a subroutine technique called Standard Call and Return. This is further described in the program listing for the Editor, but for illustrating the problem we are having, you only need to know that a D4 SEP is a call to a sub-routine. The address of that sub routine must immediatly follow the D4 byte meaning that every D4 now requires a two-byte argument. Aha! So the two bytes 0360 that follow the D4 at ML 0025 are the address of the subroutine called. And that means that the 69 at ML 0028 isn't a branch address, but an input instruction, in particular, the one needed to turn on the video. How to right such a hopelessly unassembled disassembly?

To make such things simple, all instructions needing argument bytes are kept in a table. DISASSEMBLER-7 checks the table for each instruction to see if it needs a one or two byte argument. In the program description, I will show you how to make changes to this table, but to

make simple additions of one new byte requiring two bytes of data (or to delete such an instruction), I have included a routine that allows you to add one byte to this table.

First, using Key D, select a SHOW FROM address 0000. Note that 0000 contains a 91 instruction. Press Key 0. Write the address 0000 then the byte D4 and press Key E. D4 is now at ML 0000/ Press Key A. An exclamation point comes up, the display rolls and something wierd has happened at ML 0000. Before going further, change ML 0000 back to 91 using Key 0.

Now SHOW FROM ML 001F again, and check ML 0025 against the typed listing. The D4 is correctly read as a call to the sub routine at ML 0330 and the 69 has also been correctly disassembled as an input instruction.

I wish that were the end of the problem, but it isn't. In the case where you have a block of pure data, DISASSEMBLER-7 will not be able to tell it from instructions. This is not so bad as long as neither of the last two bytes of data resemble instructions needing two or one byte arguments. If they do, the program may pick up the next bytes in your program and treat them as arguments. Knowing where the blocks of data are supposed to be will greatly aid you in viewing the listing.

With the ability to add your own mnemonics

(view ML's 0800-09FF using TEXT EDITOR-21), write to memory, tape input/output, and view disassembled programs with versatility and ease, DISASSEMBLER-7 should be a valuable addition to your software library. I hope you enjoy using the program, and find it as useful as I have. Good luck with your programming!

REGISTER ASSIGNMENT

R0 DMA Pointer
R1 Interrupt Program Counter
R2 Stack pointer @ 00FF
R3 Program counter; all routines except where noted
R4 Dedicated pointer to Call routine
R5 Dedicated pointer to Return routine
R6 Pointer to return and arguments
R7 Display cursor (invisible)/Utility in disassemble sub
R8 R8.0 is timer in interrupt; R8.1 hold key pressed and ASCII digits
R9 Pointer to text buffer - one page fixed at 0400
RA Pointer to data for disassembly @ 0500-06FF
RB RB.0 is for utility; RB.1 is display page address
RC PC for keyboard scan/pointer to ASCII mnemonic strings
RD Pseudo address holder (of data for disassembly); User set or 0000 start
RE Utility; loops, data passing, etc.
RF Utility; loops, data passing, etc.

MAIN PROGRAM

0000-002E Initialization
002F-0042 Main loop - function decoding
0050-0061 Control function jump table
0062-0075 Page forward routine with wrap-around. Not a sub
0076-008E Function calls and loop returns
0090-00D3 Tape read/write subroutine. Do not call directly.

RESERVED MEMORY

00E0-00FF Stack
0400-04FF Text buffer - initialization not needed
0500-07FF Data buffer. Enter program for disassembly. Three pages maximum

0800-0995 Mnemonic look-up table. See instructions to modify.
09C0-09E7 Argument look-up table. See instructions to modify.
0A00-0BFF ASCII Character set. See CHARACTER DESIGNER
0C00-0FFF 4-page display refresh

SUBROUTINE LIST

0100-0145 Disassemble subroutine. Orders disassembly subs, controls data pointers
0146-016C Enter new SHOW FROM address
016F-018F Displacement calculation. RF.0 indicates selected address too big or too small
0190-01AE Enter new START FROM (pseudo) address
01AF-01DB Writy byte @ address
01DC-01F9 Tape read/write; control routine

0200-0210 Print address and instruction; 2nd entry @ 020B for arguments
0211-021E ASCII conversion. Hex digits in RE.0 converted to ASCII
021F-0235 Convert/store ASCII in text. For 2 digit bytes in RF.1
0236-0252 Index RC to ASCII string; looks up mnemonics in table
0253-0291 Special cases decoding. Prepares instruction for look-up table
0292-02AE Test for arguments. RB.0 indicates number required.
*1 02AF-02FA Get/Display keyboard entries. # passed by caller; 2 or 4 entries.

0300-0311 4-page interrupt routine with R8.0 timer
0312-0321 Call routine running in R4
0322-032E Return routine running in R5
0330-0341 Erase display pages (4)
*2 0342-036B Home cursor. Resets R7, R9 to top left positions
036C-0394 Display character. Does one ASCII character in R8.1 @ R7
0395-03AB Display bit row. Does one row for character, XOR for right characters.
03AC-03B9 Clear text (Display) buffer. Sets 0400-04FF to ASCII spaces (20)
03BA-03D2 Adjust RA, RD for arguments. Avoids cutting into data.
03D3-03E3 Error Message
03E4-03ED Position R7, R9 for displaying keyboard entries.
03EE-03FF Add table entry. Adds a 2-byte argument to table.

---

*1 02AF-02BF Function display -- displays messages passed by caller
02C0-02FA Get / Display.....etc.

*2 0342-0361 Display text -- causes contents @ 0400 to be displayed on screen
0362-036B Home cursor.....etc.

INITIALIZATION

```
0000   91    GHI  R1    ;Last RAM page
  01   FF    SMI        ;Subtract 3
  02   03
  03   BB    PHI  RB    ;RB = top of 4 page display
  04   F8    LDI
  05   03
  06   B1    PHI  R1    ;R1.1 = 03
  07   B4    PHI  R4    ;R4.1 = 03
  08   B5    PHI  R5    ;R5.1 = 03
  09   F8    LDI
  0A   FF
  0B   A2    PLO  R2    ;R2.0 = FF (Stack)
  0C   F8    LDI
  0D   02
  0E   A1    PLO  R1    ;R1 = 0302 Interrupt routine
  0F   F8    LDI
0010   13
  11   A4    PLO  R4    ;R4 = 0313 Call routine
  12   F8    LDI
  13   23
  14   A5    PLO  R5    ;R5 = 0323 Return routine
  15   90    GHI  R0    ;(=00)
  16   B2    PHI  R2    ;R2 = 00FF (Stack)
  17   A9    PLO  R9    ;R9.0 = 00
  18   B3    PHI  R3    ;R3.1 = 00 Prepare for use as PC
  19   AA    PLO  RA    ;RA.0 = 00
  1A   F8    LDI
  1B   04
  1C   B9    PHI  R9    ;R9 = 0400 Fixed text page pointer
  1D   F8    LDI
  1E   05
  1F   BA    PHI  RA    ;RA = 0500 Data pointer
0020   F8    LDI
  21   24
  22   A3    PLO  R3    ;R3 = 0024
  23   D3    SEP  R3    ;R3 = PC
  24   E2    SEX  R2    ;X = 2
  25   D4    SEP  R4
  26   03               ;Call erase display pages
  27   30
  28   69    INP        ;Video on
  29   93    GHI  R3    ;(00)
  2A   BD    PHI  RD
  2B   AD    PLO  RD    ;RD = 0000 Pseudo address holder
  2C   D4    SEP  R4
  2D   01               ;Call Tape Read - Initial data loading
  2E   DC
```

MAIN LOOP - FUNCTION DECODING

```
002F    D4    SEP   R4    ;Begin Main Loop
  30    01                ;Call Disassemble
  31    00
  32    D4    SEP   R4
  33    03                ;Call Display Text page
  34    42
  35    F8    LDI
  36    81
  37    BC    PHI   RC
  38    F8    LDI
  39    95
  3A    AC    PLO   RC    ;RC = 8195 For Keyboard Scan in ROM
  3B    DC    SEP   RC    ;Call Key Scan - Key Press in D on return
  3C    F9    ORI         ;"OR" with 50 hex to create address
  3D    50                ;For jump table
  3E    AC    PLO   RC
  3F    93    GHI   R3
0040    BC    PHI   RC    ;Set RC=Jump table address on this page @ 0050-005F
  41    0C    LDN   RC    ;Get data from jump table
  42    A3    PLO   R3    ;Put in R3.0 to jump to function selected
```

CONTROL FUNCTION JUMP TABLE

```
0050    80                ;Enter (write) byte @ address
  51    85                ;Tape Read/Write
  52    60
  53    60
  54    60
  55    60
  56    60
  57    60
  58    60
  59    60
  5A    8A                ;Add table entry
  5B    60
  5C    62                ;Page forward
  5D    76                ;Enter Show From address
  5E    7B                ;Enter Start From address
  5F    60
0060    30                ;Branch to 0035
  61    35                ;For next key press (notice all unused table entries
                           =60)
```

PAGE FORWARD ROUTINE WITH WRAP-AROUND

```
0062   9A   GHI  RA
  63   FB   XRI       ;Test if RA data pointer past last byte
  64   08
  65   3A   BNZ       ;If not, branch to exit and continue
  66   2F              ;@ 002F
  67   9A   GHI  RA   ;Subtract 03 from both RD.1 and RA.1
  68   FF   SMI
  69   03              ;This assures both pointers will
  6A   BA   PHI  RA
  6B   9D   GHI  RD   ;Be accurately set even when the
  6C   FF   SMI       ;Last byte required an argument (ie "F8")
  6D   03
  6E   BD   PHI  RD
  6F   8A   GLO  RA   ;Then decrement both pointers
0070   32   BZ
  71   2F              ;To assure start from data @
  72   2A   DEC  RA
  73   2D   DEC  RD   ;0500 and not 0501 or 0502 which could
  74   30   BN        ;Happen again if last byte required an
  75   6F              ;Argument (exit routine @ 0070)
```

FUNCTION CALLS AND LOOP RETURNS

```
0076   D4   SEP  R4
  77   01              ;Call enter new Show From address
  78   46
  79   30   BN        ;Branch back to main loop
  7A   2F
  7B   D4   SEP  R4
  7C   01              ;Call enter new Start From address
  7D   90
  7E   30   BN
  7F   2F
0080   D4   SEP  R4
  81   01              ;Call Write Byte at address
  82   AF
  83   30   BN
  84   2F
  85   D4   SEP  R4
  86   01              ;Call Tape Read/Write - control routine
  87   DC
  88   30   BN        ;Branch back to Main Loop
  89   2F
  8A   D4   SEP  R4
  8B   03              ;Call Add table entry
  8C   EE
  8D   30   BN
  8E   2F
  8F   00   (Filler)
```

TAPE READ/WRITE

```
0090  E2  SEX  2     ;X = 2
  91  96  GHI  R6
  92  73  STXD      ;Push R6.1
  93  86  GLO  R6
  94  73  STXD      ;Push R6.0 (Save old R6)
  95  F8  LDI
  96  05
  97  BA  PHI  RA
  98  B6  PHI  R6
  99  F8  LDI       ;Set R6 = 0500, beginning data area
  9A  00            ;Also setting RA for later display
  9B  AA  PLO  RA
  9C  A6  PLO  R6   ;And disassemble
  9D  F8  LDI
  9E  03
  9F  AE  PLO  RE   ;Set RE.0 = Number pages fixed at 3
00A0  22  DEC  R2   ;R2 - 1 to compensate for next unwanted inc.
  A1  61  INP       ;Turn off video (R(x)+1)
  A2  83  GLO  R3
  A3  A4  PLO  R4
  A4  24  DEC  R4   ;Swap program counters - (R4=PC) R4 must
  A5  93  GHI  R3       later be set = 0313 for sub routine calls
  A6  B4  PHI  R4
  A7  D4  SEP  R4
  A8  F8  LDI
  A9  80
  AA  B3  PHI  R3   ;R3.1 = Read/Write page location in ROM
  AB  8F  GLO  RF
  AC  FB  XRI       ;Test RF.0 if Write selected
  AD  0F
  AE  3A  BNZ       ;If not, branch to test for Read
  AF  B6
00B0  F8  LDI       ;Begin call to Tape Write
  B1  91
  B2  A3  PLO  R3   ;Set R3 = Write sub in ROM @ 8091
  B3  D3  SEP  R3   ;Do sub - Write 3 pages to tape
  B4  30  BN        ;Branch to exit
  B5  C1
  B6  8F  GLO  RF
  B7  FB  XRI       ;Test if Tape Read selected (RF.0 = 0B)
  B8  0B
  B9  3A  BNZ       ;If not, branch to exit, turning video
  BA  C0            ;Back on
  BB  F8  LDI       ;Begin call to Tape Read
  BC  C2
  BD  A3  PLO  R3   ;R3= 0BC2 - Read routine in ROM
  BE  D3  SEP  R3   ;Do sub - Read 3 pages
  BF  38  SKP       ;Always skip
```

```
00C0   69   OUT        ;Turn on video when cancelling Read/Write operation
  C1   E2   SEX  2     ;X = 2 Begin exit
  C2   60   IRX        ;Point to saved R6
  C3   72   LDXA       ;Pop R6.0
  C4   A6   PLO  R6    ;Restore R6.0
  C5   F0   LDX        ;Pop R6.1
  C6   B6   PHI  R6    ;Restore R6.1
  C7   84   GLO  R4
  C8   A3   PLO  R3
  C9   23   DEC  R3    ;Swap PC's (R3 = PC)
  CA   94   GHI  R4
  CB   B3   PHI  R3
  CC   D3   SEP  R3
  CD   F8   LDI
  CE   03
  CF   B4   PHI  R4    ;Reset R4 = 0313 for use as
00D0   F8   LDI
  D1   13              ;Dedicated call routine PC
  D2   A4   PLO  R4
  D3   D5   SEP  R5    ;Return

                          DISASSEMBLE SUB

0100   D4   SEP  R4
  01   03              ;Call home cursor to reset R7, R9 -(R7
  02   62              ;however has other function here)
  03   D4   SEP  R4
  04   03              ;Call Clear Text buffer to prepare
  05   AC              ;For new text
  06   F8   LDI
  07   10
  08   A7   PLO  R7    ;R7.0 = Loop count (10 hex = 16 lines per display)
  09   D4   SEP  R4
  0A   02              ;Call Print Address and Instruction-Done initially for
  0B   00              ;Each new line
  0C   1D   INC  RD    ;RD + 01 (Pseudo address holder)
  0D   D4   SEP  R4
  0E   02              ;Call Special Cases - corrected (if needing
  0F   53              ;Correction) value passed back in R7.1
0110   D4   SEP  R4
  11   02              ;Call Index RC to mnemonic ASCII string
  12   36
  13   D4   SEP  R4
  14   02              ;Call agruments test - number of data bytes
  15   92              ;To be printed indicated in RB.0
  16   8B   GLO  RB
  17   32   BZ         ;If RB.0 = 00, then no arguments need
  18   21              ;Printing, exit passing next routine
  19   1A   INC  RA    ;Data pointer + 01 - next data byte
```

```
011A   1D   INC  RD   ;Pseudo address holder + 01 to account for data
  1B   D4   SEP  R4                                              printed
  1C   02             ;Call Print data - 2nd entry of print
  1D   0B             ;Address/Instruction sub
  1E   2B   DEC  RB   ;RB.0 - 01
  1F   30   BN
0120   16             ;Loop until RB.0 goes to zero (RA will point to last
  21   29   DEC  R9   ;R9 - 1 -Decrement data pointer's       data printed)
  22   89   GLO  R9
  23   FA   ANI       ;Last 4 bits equal zero
  24   0F
  25   3A   BNZ       ;Then R9 points to beginning of
  26   21             ;Line again
  27   89   GLO  R9
  28   FC   ADI       ;Add 0C (12 decimal) to R9
  29   0C
  2A   A9   PLO  R9   ;R9 points to text area for mnemonic
  2B   4C   LDA  RC   ;Get ASCII character from mnemonic string
  2C   32   BZ        ;If = 00 (null) then done
  2D   32             ;Exit
  2E   59   STR  R9   ;Else store the character
  2F   19   INC  R9   ;In text, increment text pointer
0130   30   BN        ;And loop until done
  31   2B
  32   89   GLO  R9   ;Test last four bits R9
  33   FA   ANI
  34   0F             ;When = 00, then R9 points to next
  35   32   BZ
  36   3A             ;Line
  37   19   INC  R9   ;Else continue to increment R9
  38   30   BN        ;Until this becomes true and the carriage
  39   32             ;Return/Line feed is complete
  3A   1A   INC  RA   ;RA + 1 Data pointer to next byte for disassembly
  3B   E2
  3C   9A   GHI  RA
  3D   FB   XRI       ;Test RA.1; when = 08, then past
  3E   08             ;The end of data for disassembly
  3F   32   BZ
0140   45             ;Therefore exit
  41   27   DEC  R7   ;Else decrement loop
  42   87   GLO  R7   ;Counter and test for R7.0 = 00
  43   3A   BNZ       ;If not,
  44   09             ;Loop until done to 0109
  45   D5   SEP  R5   ;Return
```

ENTER NEW SHOW FROM ADDRESS

```
0146   D4    SEP  R4
  47   02              ;Call Function display
  48   AF
  49   53 48 4F 57 20 46 52 4F 4D 00 - Text ("Show From")
0153   D4    SEP  R4
  54   03              ;Call position R7 R9 for keyboard entries
  55   E4
  56   D4    SEP  R4
  57   02              ;Call keyboard entry - data in RF
  58   C0
  59   04              ;Pass loop value for 4 key presses
  5A   D4    SEP  R4
  5B   01              ;Call Displacement calculation
  5C   6F
  5D   8F    GLO  RF   ;Test error flag RF.0
  5E   3A    BNZ       ;Branch if set to error
  5F   64
0160   D4    SEP  R4
  61   03              ;Else call Adjust RA RD to avoid cutting
  62   BA              ;Into data byte
  63   D5    SEP  R5   ;Return
  64   D4    SEP  R4   ;Call Error message -(Branch to here for error retry
  65   03                                                 from other subs)
  66   D3
  67   F8    LDI       ;Then set RA RD to first
  68   00
  69   AD    PLO  RD   ;Byte of whatever page they are on
  6A   AA    PLO  RA
  6B   30    BN        ;Branch to further adjust RA RD and return
  6C   60
  6D-6E Filler
```

DISPLACEMENT CALCULATION

```
016F   E2    SEX  2    ;X = 2
  70   9D    GHI  RD   ;Get high part pseudo address
  71   52    STR  R2   ;Push
  72   9F    GHI  RF   ;Get high part new selected address
  73   F7    SM        ;Subtract RF.1 - RD.1 for displacement
  74   52    STR  R2   ;Push result (in 2's compliment form)
  75   9A    GHI  RA
  76   F4    ADD       ;Add displacement to RA.1
  77   52    STR  R2   ;Push new RA.1
  78   FD    SDI       ;Subtract 07 - RA.1 (If negative, RA.1 would be > 07)
  79   07
  7A   3B    BM        ;Branch if negative to error
  7B   8C
```

```
017C   F0    LDX          ;Get new RA.1
  7D   FF    SMI          ;Subtract RA.1 - 05 (If negative, RA.1 would be < 05)
  7E   05
  7F   3B    BM           ;Branch if negative to error
0180   8C
  81   F0    LDX          ;Else get RA.1 (new) value -(Tested to be in bounds)
  82   BA    PHI  RA      ;Put in RA.1
  83   8F    GLO  RF      ;Get RF.0 part of new address
  84   AA    PLO  RA      ;Put in RA.0
  85   AD    PLO  RD      ;Put in RD.0
  86   9F    GHI  RF      ;Get RF.1 part of new address
  87   BD    PHI  RD      ;Put in RD.1
  88   F8    LDI          ;Store 00 in RF.0 to indicate RA was
  89   00                 ;Calculated to be in bounds
  8A   AF    PLO  RF
  8B   D5    SEP  R5      ;Return
  8C   F8    LDI          ;Store 01 in RF.0 to indicate RA was
  8D   01                 ;Not in bounds - (Error flag)
  8E   AF    PLO  RF
  8F   D5    SEP  R5      ;Return

                          ENTER NEW PSEUDO ADDRESS

0190   D4    SEP  R4
  91   02                 ;Call Function display
  92   AF
  93   53 54 41 52 54 20 46 52 4F 4D 00 - Text ("Start From")
019E   D4    SEP  R4
  9F   03                 ;Call position R7 R9 for keyboard entries
01A0   E4
  A1   D4    SEP  R4
  A2   02                 ;Call Keyboard entry - Data in RF
  A3   C0                 ;  "     "        "     "   "  "
  A4   04                 ;Pass loop value (4 digits)
  A5   9F    GHI  RF      ;Transfer new pseudo start address to RD
  A6   BD    PHI  RD      ;  "       "    "     "       "     "  "
  A7   F8    LDI
  A8   05                 ;Then reset RA data pointer to data beginning
  A9   BA    PHI  RA      ;  "    "   "    "    "     "    "     "
  AA   F8    LDI          ;  "    "   "    "    "     "    "     "
  AB   00                 ;And limit new pseudo start to
  AC   AD    PLO  RD      ;A page beginning (RD.0 = 00)
  AD   AA    PLO  RA
  AE   D5    SEP  R5      ;Return
```

WRITE BYTE @ ADDRESS

```
01AF   D4    SEP  R4
  B0   02              ;Call Function display
  B1   AF
  B2   57 52 49 54 45 20 42 59 54 45 00   ("Write Byte")
  BD   D4    SEP  R4
  BE   03              ;Call Position R7 R9 for keyboard entry
  BF   E4
01C0   D4    SEP  R4
  C1   02              ;Call Keyboard entry
  C2   C0
  C3   04              ;Pass loop value - 4 for an address
  C4   D4    SEP  R4
  C5   01              ;Call Displacement/RA boundary check
  C6   6F              ;(RF = 00 = yes  RF = 01 no, not in bounds)
  C7   8F    GLO  RF
  C8   3A    BNZ       ;If RA not in bounds, branch to Error-Retry
  C9   64              ;In Show From sub @ 0164
  CA   19    INC  R9   ;R9 + 1 to create space after displayed address
  CB   D4    SEP  R4
  CC   02              ;Call Keyboard entry (Answer in RF.1)
  CD   C0
  CE   02              ;Pass loop value - 2 for a single byte
  CF   9F    GHI  RF
01D0   BE    PHI  RE   ;Store byte in temporary register RE.1
  D1   DC    SEP  RC   ;Do keyboard scan
  D2   FB    XRI
  D3   0E              ;Test if key "E" prassed, if not,
  D4   3A    BNZ       ;Branch to recycle for new entry @ 01AF.
  D5   AF              ;(Could branch to exit)
  D6   9E    GHI  RE   ;Get the saved byte
  D7   5A    STR  RA   ;Store in data area
  D8   D4    SEP  R4
  D9   03              ;Call Adjust RA RD to avoid displaying in
  DA   BA              ;The middle of a data byte
  DB   D5    SEP  R5   ;Return
```

TAPE READ/WRITE - CONTROL ROUTINE

```
01DC   D4    SEP  R4
  DD   02              ;Call Function display
  DE   AF
  DF   52 45 41 44 2F 57 52 49 54 45 3F 00   ("Read/Write?")
01EB   F8    LDI
  EC   81              ;Initialize RC to Key scan ROM routine
  ED   BC    PHI  RC
  EE   F8    LDI       ;Initialize RC to Key scan ROM routine
  EF   95
```

```
01F0   AC   PLO  RC   ;Initialize RC to Key scan ROM routine @ 8195
  F1   DC   SEP  RC   ;Do Key scan
  F2   AF   PLO  RF   ;Store instruction in RF.0
  F3   D4   SEP  R4
  F4   00             ;Call Tape Read/Write sub
  F5   90
  F6   F8   LDI       ;Load RD.0 with 00 to assure
  F7   00
  F8   AD   PLO  RD   ;Start (pseudo address) from page beginning
  F9   D5   SEP  R5   ;Return

                  PRINT ADDRESS AND INSTRUCTION

0200   9D   GHI  RD   ;Get first half pseudo address in RD.1
  01   BF   PHI  RF   ;Put in RF.1 to pass to sub
  02   D4   SEP  R4
  03   02             ;Call Convert/Store ASCII in text
  04   1F
  05   8D   GLO  RD   ;Get second half pseudo address in RD.0
  06   BF   PHI  RF   ;Put in RF.1 to pass to sub
  07   D4   SEP  R4
  08   02             ;Call Convert/Store ASCII in text
  09   1F             ;(Address is now printed)
  0A   19   INC  R9   ;Buffer pointer + 01 to create space
 *0B   0A   LDN  RA   ;Get instruction for disassembly
  0C   BF   PHI  RF   ;Put in RF.1 to pass to sub
  0D   D4   SEP  R4
  0E   02             ;Call Convert/Store ASCII in text
  0F   1F
0210   D5   SEP  R5   ;Return (R9 points just after last instruction printed)

                  ASCII CONVERSION (HEX DIGITS)

0211   8E   GLO  RE   ;Get value passed by caller
  12   FD   SDI       ;Value - 09 (If result negative, value > 9)
  13   09
  14   33   BPZ       ;Branch if value ≤ 9 (not a letter)
  15   1A
  16   8E   GLO  RE   ;Get the same value
  17   FC   ADI       ;Add 07 if value is a letter
  18   07
  19   AE   PLO  RE   ;Store in RE.0
  1A   8E   GLO  RE   ;Get value in RE.0 (Either plus 07 or not depending)
  1B   FC   ADI       ;Add 30 hex always to complete conversion
  1C   30
  1D   AE   PLO  RE   ;Store converted value in RE.0
  1E   D5   SEP  R5   ;Return
```

*Note- Entry point for printing arguments to instructions

```
                        CONVERT/STORE ASCII IN TEXT

021F   9F   GHI  RF    ;Get byte for conversion passed in RF.1
0220   F6   SHR         ;Shift first digit to right
  21   F6   SHR         ;  "      "      "    "    "
  22   F6   SHR         ;  "      "      "    "    "
  23   F6   SHR         ;  "      "      "    "    "
  24   AE   PLO  RE    ;Put in RE.0 for passing to sub
  25   D4   SEP  R4
  26   02               ;Call ASCII conversion (Returns ASCII
  27   11               ;In RE.0)
  28   8E   GLO  RE    ;Get the ASCII conversion
  29   59   STR  R9    ;Store in text at R9 (First digit)
  2A   19   INC  R9    ;R9 + 01
  2B   9F   GHI  RF    ;Get the same byte for conversion
  2C   FA   ANI        ;"AND" with 0F for second digit
  2D   0F
  2E   AE   PLO  RE    ;Put in RE.0 for passing to sub
  2F   D4   SEP  R4
0230   02               ;Call ASCII conversion
  31   11
  32   8E   GLO  RE    ;Get the ASCII conversion
  33   59   STR  R9    ;Store in text at R9 (Second digit)
  34   19   INC  R9    ;R9 + 01
  35   D5   SEP  R5    ;Return

                        INDEX RC TO ASCII STRING

0236   F8   LDI
  37   08
  38   BC   PHI  RC
  39   F8   LDI         ;Initialize RC to point to
  3A   00
  3B   AC   PLO  RC    ;Beginning mnemonic table @ 0800
  3C   E2   SEX  2     ;X = 2
  3D   97   GHI  R7    ;Get the decoded instruction in R7.1
  3E   52   STR  R2    ;Push for upcoming comparison
  3F   4C   LDA  RC    ;Get byte from a table entry
0240   F3   XOR         ;Compare with byte on stack
  41   3A   BNZ
  42   44               ;If not equal, continue
  43   D5   SEP  R5    ;Else return -- RC points to instruction mnemonic
  44   4C   LDA  RC
  45   3A   BNZ         ;Increment RC to next table entry by
  46   44               ;Advancing, testing for null (00) byte
  47   9C   GHI  RC
  48   FB   XRI         ;Test if RC went too far -- past table end
  49   0A               ;(00 byte at 09FF prevents runaway search)
  4A   3A   BNZ
```

```
024B  3F                ;If not continue
 4C   F8   LDI          ;Else load RC
 4D   09
 4E   BC   PHI  RC      ;With error message (???)
 4F   F8   LDI
0250  93                ;Address
 51   AC   PLO  RC
 52   D5   SEP  R5      ;And return

                         SPECIAL CASES DECODING

0253  E2   SEX  2       ;X = 2
 54   0A   LDN  RA      ;Get the instruction for disassembly
 55   32   BZ           ;If = 00 (IDL) Branch to exit @ 028F
 56   8F
 57   FA   ANI
 58   F0                ;Strip second digit to test for "6N" In/Out bytes
 59   FB   XRI
 5A   60
 5B   3A   BNZ          ;If not a 6N type, branch to next test
 5C   73
 5D   0A   LDN  RA      ;Get the instruction
 5E   FB   XRI
 5F   60                ;If 6N type is 60, IRX, branch
0260  32   BZ
 61   8F                ;To exit @ 028F
 62   0A   LDN  RA
 63   FB   XRI
 64   68                ;If 6N type is an illegal code, (68)
 65   32   BZ
 66   8F                ;Branch to exit @ 028F (Error will be output to text)
 67   0A   LDN  RA
 68   FD   SDI          ;Subtract -- 67 - instruction (If > 67, then
 69   67                ;Result negative, if = or < 67, then result positive)
 6A   33   BPZ          ;Branch if positive to set X of NX = F
 6B   8A                ;(6N type is an out instruction)
 6C   0A   LDN  RA      ;Get instruction (now determined to be an "INP")
 6D   FA   ANI
 6E   F0                ;Strip the second digit
 6F   F9   ORI
0270  01                ;OR with 01 to put a 1 in second digit
 71   B7   PHI  R7      ;Store in R7.1 for look up table
 72   D5   SEP  R5      ;Return
 73   0A   LDN  RA      ;Get instruction for disassembly
 74   FA   ANI
 75   F0                ;Strip the second digit
 76   52   STR  R2      ;Push for comparison tests
 77   FB   XRI
 78   30                ;If = 30 (3N type)
```

```
0279   32   BZ          ;Branch to exit - no conversion needed
  7A   8F
  7B   F0   LDX         ;Get the same instruction from stack
  7C   FB   XRI
  7D   70               ;If = 70 (7N type)
  7E   32   BZ          ;Branch to exit
  7F   8F
0280   F0   LDX
  81   FB   XRI
  82   C0               ;If = C0 (CN type)
  83   32   BZ          ;Branch to exit
  84   8F
  85   F0   LDX
  86   FB   XRI
  87   F0               ;If = F0 (FN type)
  88   32   BZ          ;Branch to exit
  89   8F
  8A   0A   LDN  RA     ;Else get the instruction
  8B   F9   ORI
  8C   0F               ;"OR" to cnnvert to XF for look up table
  8D   B7   PHI  R7     ;Store in R7.1
  8E   D5   SEP  R5     ;Return
  8F   0A   LDN  RA     ;Some tests branch to here, loading R7.1
0290   B7   STR  R7     ;With the unconverted instruction
  91   D5   SEP  R5     ;And returning

                          TEST FOR ARGUMENTS

0292   F8   LDI
  93   01
  94   AB   PLO  RB     ;Initialize RB.0 = 01 -Argument count
  95   F8   LDI
  96   02
  97   AF   PLO  RF     ;RF.0 = Loop count - 2 part table
  98   0A   LDN  RA     ;Get instruction for disassembly
  99   E2   SEX  2      ;X = 2
  9A   52   STR  R2     ;Push value for comparison test
  9B   F8   LDI
  9C   09
  9D   BE   PHI  RE     ;Set RE = beginning
  9E   F8   LDI
  9F   C0               ;Of argument table @ 09C0
02A0   AE   PLO  RE
  A1   4E   LDA  RE     ;Get entry from table - advance pointer
  A2   32   BZ          ;If 00 (null) found (end of part "N" table)
  A3   A8               ;Branch to done test
  A4   F3   XOR         ;Else compare with stacked instruction
  A5   3A   BNZ         ;If not equal, loop till equality or
  A6   A1               ;End of table part "N" found
```

```
02A7   D5    SEP   R5    ;Return, number arguments in RB.0
  A8   1B    INC   RB    ;RB.0 + 1 for second part table (Last INC ignored)
  A9   2F    DEC   RF    ;Loop counter RF.0 - 01
  AA   8F    GLO   RF
  AB   3A    BNZ         ;Loop through table till done or equality found
  AC   A1
  AD   AB    PLO   RB    ;Else set RB.0 = 00, no arguments required
  AE   D5    SEP   R5    ;Return

                         FUNCTION DISPLAY

02AF   D4    SEP   R4
  B0   03                ;Call Home cursor (sets R7 R9)
  B1   62
  B2   D4    SEP   R4
  B3   03                ;Call Clear text buffer
  B4   AC
  B5   46    LDA   R6    ;Get text from calling routine
  B6   32    BZ          ;Test for null end of string character
  B7   BC                ;Branch if end
  B8   59    STR   R9    ;Else store in text
  B9   19    INC   R9    ;Advance pointer to text
  BA   30    BN          ;Loop till done
  BB   B5
  BC   D4    SEP   R4
  BD   03                ;Call Display text
  BE   42
  BF   D5    SEP   R5    ;Return

                    GET/DISPLAY KEYBOARD ENTRIES

02C0   E2    SEX   2     ;X = 2
  C1   F8    LDI
  C2   FF
  C3   73    STXD        ;Store FF stop byte on stack
  C4   46    LDA   R6    ;Get # entries wanted from calling routine
  C5   AB    PLO   RB    ;Put in loop counter RB.0 (1-4 possible)
  C6   F8    LDI
  C7   81                ;Initialize RC to ROM Key scan routine
  C8   BC    PHI   RC
  C9   F8    LDI         ;Initialize RC to ROM Key scan routine
  CA   95
  CB   AC    PLO   RC    ;Initialize RC to ROM Key scan routine
  CC   DC    SEP   RC    ;Get keyboard entry
  CD   AE    PLO   RE    ;Put in RE to pass to sub
  CE   73    STXD        ;(Byte is already there, but this decrements the
  CF   D4    SEP   R4                                              pointer)
02D0   02                ;Call ASCII conversion - result in RE.0
  D1   11
```

```
02D2   8E   GLO  RE   ;Get the converted byte
  D3   59   STR  R9   ;Store in text for display
  D4   B8   PHI  R8   ;Also put in R8.1 for passing to sub
  D5   D4   SEP  R4
  D6   03             ;Call Display Character @ R7
  D7   6C
  D8   19   INC  R9   ;Point to next character slot
  D9   89   GLO  R9
  DA   F6   SHR       ;Shift R9.0 right to test if even or odd
  DB   33   BDF
  DC   DE             ;If odd, (DF=01), skip the INC R7 instruction
  DD   17   INC  R7   ;R7 + 1 - Next display byte (there are 2 char. per
  DE   2B   DEC  RB   ;Loop count - 1                                byte)
  DF   8B   GLO  RB
02E0   3A   BNZ       ;Loop till all entries processed and displayed
  E1   C6             ;All entries on stack ending with "FF" stop  byte
  E2   E2   SEX  2    ;X = 2
  E3   F8   LDI       ;Set RF.1 (answer) = 00
  E4   00             ; "   "     "       "  "
  E5   BF   PHI  RF   ; "   "     "       "  "
  E6   60   IRX
  E7   9F   GHI  RF
  E8   AF   PLO  RF   ;Transfer contents RF.1 into RF.0
  E9   72   LDXA      ;Pop and advance pointer (2nd ½ byte)
  EA   BF   PHI  RF   ;Store in RF.1
  EB   F0   LDX       ;Pop next value (first ½ byte)
  EC   FE   SHL       ;Shift left X 4 so digit in left position
  ED   FE   SHL
  EE   FE   SHL
  EF   FE   SHL
02F0   52   STR  R2   ;Re-stack shifted byte
  F1   9F   GHI  RF
  F2   F1   OR        ;OR existing RF.1 with byte on stack in
  F3   BF   PHI  RF   ;Order to combine
  F4   60   IRX
  F5   F0   LDX       ;Test next byte on stack for FF stop byte
  F6   FB   XRI
  F7   FF
  F8   3A   BNZ       ;If not detected, loop for next byte data
  F9   E7
  FA   D5   SEP  R5   ;Else return, stack "right", byte(s) in RF
```

```
0300
  ↓       4-Page Interrupt
0311

0312
  ↓       Call Routine                    See Text Editor-21 for
0321                                      Full Description

0322
  ↓       Return Routine
032E
```

ERASE DISPLAY PAGES

```
0330  9B   GHI  RB
  31  BF   PHI  RF    ;RF.1 = RB.1 - Top display page
  32  FC   ADI
  33  04             ;Add 4 to value to indicate
  34  E2   SEX  2     ;When pointer goes past last display page
  35  52   STR  R2    ;Push this value onto stack
  36  F8   LDI
  37  00             ;Store 00 in RF.0
  38  AF   PLO  RF    ;To point to first display byte
  39  F8   LDI
  3A  00             ;Get 00 for erasing
  3B  5F   STR  RF    ;Store in Display area
  3C  1F   INC  RF    ;RF + 01
  3D  9F   GHI  RF
  3E  F3   XOR        ;Test if RF.1 = stacked byte
  3F  3A   BNZ
0340  39             ;If not, loop till done
  41  D5   SEP  R5    ;Else, return
```

DISPLAY TEXT

```
0342  D4   SEP  R4
  43  03             ;Call Home cursor (R7 & R9 reset)
  44  62
  45  09   LDN  R9    ;Get a character
  46  B8   PHI  R8    ;R8.1 holds ASCII character
  47  D4   SEP  R4
  48  03             ;Call Display a character @ R7
  49  6C
  4A  19   INC  R9    ;Point to next character
  4B  89   GLO  R9    ;Test R9.0 for Even/Odd
  4C  F6   SHR        ;Shift LSB into DF
  4D  33   BDF        ;Branch if odd (DF=01) to display next character
  4E  45             ;@ R7 (does 2 characters per display byte)
  4F  17   INC  R7    ;Cursor + 01
```

```
0350   87   GLO  R7
  51   FA   ANI       ;Test if last 3 bits R7 = 00 by masking out
  52   07             ;All others
  53   3A   BNZ       ;If not, continue to display
  54   45             ;Characters to the end of a line (R7.0 = X000)
  55   87   GLO  R7
  56   FC   ADI
  57   38
  58   A7   PLO  R7   ;Add 38 hex (56 decimal) to
  59   97   GHI  R7
  5A   7C   ADCI      ;Cursor for next row down
  5B   00
  5C   B7   PHI  R7
  5D   FB   XRI       ;Then test if cursor went
  5E   10
  5F   3A   BNZ       ;Off display -- if not, loop
0360   45             ;Until done
  61   D5   SEP  R5   ;Return

                          HOME CURSOR

0362   9B   GHI  RB
  63   B7   PHI  R7   ;R7.1 = RB.1
  64   F8   LDI
  65   04
  66   B9   PHI  R9   ;R9.1 = 04
  67   F8   LDI
  68   00
  69   A7   PLO  R7   ;R7 = 0C00  Top display page
  6A   A9   PLO  R9   ;R9 = 0400  Top text buffer
  6B   D5   SEP  R5   ;Return - R7 R9 reinitialized

                          DISPLAY A CHARACTER

036C   98   GHI  R8   ;R8.1 holds ASCII character passed by caller
  6D   FE   SHL
  6E   FE   SHL       ;Multiply (by shifting) times 04
  6F   AF   PLO  RF   ;And place in RF.0
0370   F8   LDI
  71   0A             ;Add 0A hex plus possible
  72   7C   ADCI
  73   00             ;Carry to index character bit pattern
  74   BF   PHI  RF   ;And put in RF.1
  75   F8   LDI
  76   04
  77   AE   PLO  RE   ;Set RE.0 = 04 for loop count (4 X 2 lines each char.)
  78   0F   LDN  RF   ;Get a bit pattern row (2 rows packed actually)
  79   FA   ANI
  7A   F0             ;"AND" with F0 hex for MSB's
```

```
037B  BE   PHI  RE   ;Store in temporary RE.1 to pass to display sub
  7C  D4   SEP  R4
  7D  03             ;Call Display bit row (Odd rows)
  7E  95
  7F  4F   LDA  RF   ;Get same bit pattern/Advance pointer RF
0380  FE   SHL       ;Shift left X 4 for LSB's
  81  FE   SHL       ;  "      "  "  "  "    "
  82  FE   SHL       ;  "      "  "  "  "    "
  83  FE   SHL       ;  "      "  "  "  "    "
  84  BE   PHI  RE   ;Store in temporary RE.1 to pass to display sub
  85  D4   SEP  R4
  86  03             ;Call Display bit row (Even rows)
  87  95
  88  2E   DEC  RE   ;Loop count - 01
  89  8E   GLO  RE
  8A  3A   BNZ       ;If RE.0 not = 00,
  8B  78             ;Loop until done
  8C  87   GLO  R7
  8D  FF   SMI       ;Subtract 40 hex from display cursor
  8E  40
  8F  A7   PLO  R7   ;To reset to top bit pattern row
0390  97   GHI  R7
  91  7F   SMBI       ;Take possible borrow into account
  92  00
  93  B7   PHI  R7
  94  D5   SEP  R5   ;Return

                        DISPLAY BIT ROW

0395  89   GLO  R9   ;Test data pointer R9.0
  96  F6   SHR       ;For even or odd by shifting LSB into DF
  97  3B   BNF       ;If DF = 0 (R9.0 is even) branch to skip the
  98  A1             ;Next shifting and XOR instructions
  99  E7   SEX  7    ; X = 7 (to facilitate upcoming XOR instruction)
  9A  9E   GHI  RE   ;Get the unpacked bits passed by calling routine
  9B  F6   SHR       ;Shift the bits to the right most position
  9C  F6   SHR       ;  "        "          "          "
  9D  F6   SHR       ;  "        "          "          "
  9E  F6   SHR       ;  "        "          "          "
  9F  F3   XOR       ;Exclusive OR with bits @ R7 to preserve the left
03A0  38   SKP       ;Always skip/Hand character already displayed
  A1  9E   GHI  RE   ;Get unpacked bits (only for left hand characters)
  A2  57   STR  R7   ;Store processed row in display area
  A3  87   GLO  R7
  A4  FC   ADI
  A5  08
  A6  A7   PLO  R7   ;Add 08 hex to cursor address to
  A7  97   GHI  R7
  A8  7C   ADCI      ;Point to next bit row
  A9  00
  AA  B7   PHI  R7   ;Take possible carry into account
  AB  D5   SEP  R5   ;Return
```

CLEAR TEXT (DISPLAY) BUFFER

```
03AC   F8   LDI
  AD   04
  AE   BF   PHI  RF
  AF   F8   LDI
03B0   00
  B1   AF   PLO  RF    ;Initialize RF = 0400  Text buffer beginning
  B2   F8   LDI
  B3   20              ;Get ASCII space (20)
  B4   5F   STR  RF    ;Store in text buffer
  B5   1F   INC  RF    ;RF + 1 next byte
  B6   8F   GLO  RF
  B7   3A   BNZ        ;Loop until RF.0 = 00  and one full page
  B8   B2              ;Has been set to 20's
  B9   D5   SEP  R5    ;Return
```

ADJUST RA RD FOR ARGUMENTS
(Used for scrolling & paging back to avoid cutting into data)

```
03BA   2A   DEC  RA  ;Decrement RA, RD X 02 to check first if
  BB   2A   DEC  RA   ;Byte originally pointed to is part of
  BC   2D   DEC  RD   ;A 2-byte argument
  BD   2D   DEC  RD   ;"   "      "
  BE   D4   CALL
  BF   02             ;Call Arguments test (RB.0 indicates
03C0   92             ;Match)
  C1   8B   GLO  RB   ;Test if RB.0 = 02, if so
  C2   FB   XRI       ;Then a match was found in the
  C3   02             ;2-byte argument table and RA RD are properly set
  C4   32   BZ        ;Branch to Exit
  C5   CE
  C6   1A   INC  RA   ;Else INC RA RD to test for one-byte
  C7   1D   INC  RD   ;Arguments
  C8   D4   CALL
  C9   02             ;Call Arguments test again
  CA   92
  CB   8B   GLO  RB   ;Test RB.0
  CC   32   BZ
  CD   D0             ;If = 00, then byte is not data-branch to reset
  CE   30   BN        ;To begin continue to test to be positive
  CF   BA             ;Byte jumped to cannot be data
03D0   1A             ;INC RA RD when byte is not data
  D1   1D
  D2   D5             ;Return
```

ERROR MESSAGE

```
03D3    D4    SEP  R4
  D4    02               ;Call Function display
  D5    AF
  D6    45 52 52 4F 52 2D 2D 52 45 54 52 59 00 - Text (Error--Retry)
03E3    D5               ;Return
```

POSITION R7 R9 FOR KEYBOARD ENTRIES

```
03E4    D4    SEP  R4
  E5    03               ;Call Home cursors setting R7.1 R9.1
  E6    62
  E7    F8    LDI
  E8    10
  E9    A9    PLO  R9    ;R9 = 0410 - second text line
  EA    F8    LDI
  EB    40
  EC    A7    PLO  R7    ;R7 = 0C40 - second display line
  ED    D5    SEP  R5    ;Return
```

ADD TABLE ENTRY

```
03EE    D4    SEP  R4
  EF    02               ;Call Function
03F0    AF               ;Display
  F1    21 00  - Text (!)
  F3    F8
  F4    00               ;Insure RA RD to
  F5    AA               ;Point to page tops
  F6    AD
  F7    F8
  F8    09               ;Set RC = 09E1
  F9    BC
  FA    F8               ;Last entry in
  FB    E1               ;2-byte argument
  FC    AC               ;Table
  FD    0A               ;Get byte @ RA
  FE    5C               ;Store in table
  FF    D5               ;Return
```

Editor's Note: When Tom returned the finished manuscript to us, after having proof-read it, he included the following note. Since I had no idea where in the manuscript this data would best fit, I rather arbitrarily decided to add it here.

" I mention that I will explain the configuration of the Mnemonic Look-up Table and Argument Look-up Table--but I never do.

The Mnemonic table is very simple. The hex codes for instructions precede the ASCII-encoded mnemonics, which are in turn followed by exactly one null (00) stop byte. The table, and thus the mnemonics used by the disassembler, may be changed directly by using the Text Editor to insert new mnemonics, or by selectively changing bytes using the ROM system monitor. Instructions which take the form XN (0N, 1N, 2N, etc.) are entered in the form XF; the "F" being critical to the operation of the table.

The Argument Look-up Table @ 09C0 - 09E7 is in two parts. Part one contains the hex codes for instructions which need one-byte arguments. Part two contains the codes for instructions needing two-byte arguments. The two parts are separated by a null (00) byte, and any bytes may be placed in either part to effect a change. The XF form is not used with this table.

Another serious omission is the checksum data for the Text Editor and the Disassembler. It should be included, as readers will find it very helpful."

Accordingly, Tom included a copy of the checksum data for both programs, which follows on the next page.

(See VIPER Nov. 1978 for instructions on using the following data)

64 BYTE CHECKSUM

DISASSEMBLER-7

0000-	3E	Row	DEE2	9870
		Col	8A80	5F7E
0040-	49	Row	70D7	5705
		Col	B903	9446
0080-	DD	Row	74EC	6B89
		Col	060D	F991
00C0-	00	Row	A050	0000
		Col	4AB9	750E
0100-	73	Row	DC77	CE01
		Col	F6D1	ED89
0140-	6D	Row	ED1B	82E9
		Col	FFE3	0435
0180-	83	Row	685F	DE6C
		Col	A7F7	3C60
01C0-	B6	Row	9B53	5E63
		Col	442D	326C
0200-	30	Row	F784	9E55
		Col	4070	1600
0240-	D9	Row	1927	8E7D
		Col	C3E4	CBE9
0280-	20	Row	F54C	78ED
		Col	8F45	2CB8
02C0-	28	Row	DF44	B185
		Col	20DC	8015
0300-	0F	Row	204E	5E70
		Col	98E2	A474
0340-	C4	Row	79BD	B440
		Col	DC42	5231
0380-	14	Row	6660	52EB
		Col	3662	3F7B
03C0-	20	Row	BACE	8B96
		Col	939A	1F5C

TEXT EDITOR-21

93	Row	DEE5	92FD
	Col	398E	A777
4C	Row	2296	946B
	Col	D739	E050
32	Row	1D47	DA20
	Col	251F	09F8
00	Row	0000	0000
	Col	0000	0000
FA	Row	4C6B	FBDC
	Col	DEF1	8B28
D9	Row	3463	A07E
	Col	29A7	73B0
D2	Row	FAEB	E017
	Col	D020	BA05
26	Row	26BA	515E
	Col	EDB5	D8F9
40	Row	F85A	8B35
	Col	B5E1	F9A2
C6	Row	92C0	73DD
	Col	CC4A	433D
F1	Row	F193	EE56
	Col	68D7	62D2
DA	Row	84E2	E540
	Col	8062	9E47
58	Row	204E	5CF9
	Col	C3E2	39BA
1B	Row	95B1	5B3F
	Col	7D79	FB19
B7	Row	9663	BBE1
	Col	A9AF	B4D2
04	Row	1890	0000
	Col	B8A3	0AAE

NOTE- Load the checksum program (which is relocatable) at ML 0500 after loading in either TEXT EDITOR-21 or DISASSEMBLER-7 to check the listing. Change ML 0000 to C0 05 00 to perform a long branch to the checksum program then follow the instructions in VIPER.

2) All unused memory spaces including the stack at 00E0-00FF are set to zero

3) Checksum data is not included here for either the character set or the mnemonic look up table for the disassembler. If you think there is a problem with the table, please refer to DISASSEMBLER-7's program description. The character set may be viewed or modified using my program, "A Character Designer for the VIP."

C H I P - 8 P R O G R A M E D I T O R

C H I P - 8 P R O G R A M E D I T O R

Those who have gotten past the game section in their Cosmac VIP manual, and who have begun to design their own brain teasers, will soon realize that working with the system ROM monitor has its good points and its bad points when programming in CHIP-8.

Its obvious good points are: its simplicity; the ease with which programs may be transferred to and from tape; and its value as a debugging tool. Some necessities are missing, however:

1. The ability to display an entire CHIP-8 instruction at a time. With the VIP monitor, we can see only one byte at a time, which is only half an instruction.
2. The ability to read from and write to memory without having to reset the monitor.
3. The ability to move defined blocks of code forward or backward in memory to accomodate forgotten instructions. This is a real inconvenience - especially when you have to re-enter long sections of code by hand - to make room for a single missed instruction.

Having defined the problem (which should be your first step in any programming effort), I devised an outline for a CHIP-8 program which would fill these requirements. The goal was to produce a program that could be used on "stock" VIPs (i.e., straight from the carton - no hardware modifications required). The program had to be usable with most existing CHIP-8 programs, even those presented in the manual, so it could not require alteration of the standard format of CHIP-8 programming as described in the VIP manual or the User's Guide.

The CHIP-8 Program Editor is written in machine language and resides in the first two pages of memory. It uses the last on-card memory page for display refresh. CHIP-8 programs are written and edited beginning at hex location 0200, the normal format for CHIP-8 programs. With the editor in use, you'll be able to see five CHIP-8 instructions on the screen at one time, although for ease of writing the editor, I dropped the leading zero in the address. (The leading zero serves no real purpose unless your VIP has more than 4K of memory.) Thus, location 0252 will appear on the screen as simply 252, followed by the two byte instruction or data (memory contents) stored at that address.

Instead of a cursor, the bottom screen line is designated as the "active" address. When writing to memory, the instruction being typed will automatically be entered into the address location appearing at the bottom of the display.

O P E R A T I N G I N S T R U C T I O N S

The CHIP-8 Program Editor contains only six instructions, three of which are mnemonic:

Key	Function
F	Scroll Forward. Scrolls forward one instruction (two bytes) at a time. The new byte appears at the bottom of the screen, in the "active" position, and the byte previously displayed in that position is moved up one line on the screen.
B	Scroll Backward. Scrolls backward one instruction (two bytes) at a time. The instruction in the "active" position is scrolled off the screen at the bottom, and the byte which had previously been displayed immediately above the "active" position is rolled down, becoming the "active" byte.

C Page forward. Scrolls forward five instructions at a time (fast forward scrolling).

D Page backward. Scrolls backward five instructions at a time (fast backward scrolling).

E Enter Instruction. Enters a CHIP-8 instruction into the active address displayed on the bottom line on the screen. Key E is pressed first, followed by the four-digit CHIP-8 instruction.

0 Relocate. Relocates any block of data, any length, to any new location, forward or backward, in VIP RAM.

Since the editor is written to begin at location 0200 (remember, this location will appear on the screen as "200"), it is not necessary to enter an address before inserting a new address, unlike other editors of this type. Simply use the keys to locate the desired memory location at the bottom of the screen (into the "active" position). Location 0020 contains the starting page number, and you can change this so the editor will begin at any desired page instead of at 0200 if you wish to do so.

It is not necessary to specify an address when you wish to enter code, nor is it necessary to use the VIP's reset toggle when you wish to examine other locations after entering your instructions. The editor always returns to the Read Mode, to allow you to enter a few instructions at one location, page forward, and enter instructions somewhere else, without resetting, as you must do with the VIP ROM monitor.

To Read Or Examine Memory

Using Key F, you can examine memory sequentially. With each press of the key, a new location appears on the screen in the

"active" position (the bottom line of the screen). You do not have to specify an address - just press the key.

If you wish to examine memory locations several pages away from the address currently shown in the "active" position, press Key C and scroll forward five instructions at once.

If you scroll past the desired location, it presents no problem; Key B will cause the display to scroll backward, one instruction at a time. Each time you press Key B, the address second from the bottom will be scrolled down into the "active" position.

To scroll backwards rapidly (five instructions at a time), press Key D. Each press of Key D will scroll five instructions off the "bottom" of the screen, so you can page back as far as you like very quickly.

To Write To Memory

To write to memory, press key E, then the two-byte CHIP-8 instruction. For example, press E followed by 6145, or E followed by F129. The new instruction will automatically be written into the last memory address on the screen, and the display will scroll forward by one line, to permit sequential programming.

It isn't necessary to specify an address, nor is it necessary, when you wish to examine other locations, to reset the VIP with the RUN/RESET switch after entering your instructions. The editor always returns to the Read Mode, so you can enter a few instructions at one location, page forward, and enter instructions somewhere else, without resetting (as you must do when you're using the ROM Monitor).

To Relocate Blocks Of Memory

Key 0 is the most complicated feature of the editor. The ability to move blocks of data around in memory at will is a huge timesaver! For example, suppose you have a program contained in locations 0200-0326, and discover you have left out two instructions. Worse, you find that the missing code should have been placed in locations 0250-0252. This means that all the code from 0250 to 0326 will have to be moved down four bytes to accomodate the forgotten instructions. Using the VIP ROM monitor, all this code had to be re-entered by hand - and the user's exasperation defies description! The CHIP-8 Program Editor eliminates this problem with the Key 0 function: Relocate.

First (assuming you have the editor running), write down the address of the last instruction in the block you wish to relocate. In this example, it's 326. Remember not to include the leading zero of the address! Write this address down on paper so you won't forget it. (I realize it sounds "childish" to write down the addresses you'll need to remember. I have found, while working with the editor, that it's very easy to forget one of them - so they should be written down at least until you are absolutely convinced your memory is better than mine!).

Next, locate the first address of the block to be moved; in this case, location 0250. Using the scrolling keys, bring location 250 to the "active" position on the screen.

Now mentally calculate the <u>new</u> start address of the block. In this example, we want locations 250 and 252 free so we can insert the forgotten two instructions, so the new start address will be 254. Write this address down on your sheet of paper.

With location 250 still at the bottom of the screen, press Key 0 <u>one</u> time. The editor enters a loop that sits and waits for your instructions. Now enter the address of the <u>last</u> instruction to be moved (326 in this case), and then enter the <u>new</u> start address (254) - both of which you have written down. Be careful not to enter the old start address! If

you do make an error, press Key F repeatedly. The number of times you'll have to press it will depend on where you are in the sequence, but repeated pressings of Key F will get you out of the Relocate loop and back into scroll/enter mode.

After you have entered the new start address, press Key 0 one time (this is the second time you will have pressed it). At this point, the block of data from 250 to 326 will be moved to locations 254-32A. If you press any key besides Key 0, you will automatically be returned to scroll/enter mode.

To make things easier, the editor automatically fills the vacated addresses with all zeros. You may move any size block of data to any location in VIP memory - except, of course, the first two pages (which hold the Editor Program).

The Relocate feature of the Editor Program is handy for moving subroutines around or for relocating blocks of data that may have been initially placed at the end of memory before the exact program length was known. It's also useful for closing up spaces after removing unnecessary code and for opening up spaces for insertion of forgotten code. Remember, however, that any GOTO or GOSUB instructions involving addresses within the relocated block will have to be adjusted after the code is shifted.

Summary of Key 0 Operation

1. Write down the address of the last instruction to be moved.
2. Write down the new start address.

Remember to exclude the leading zero from these addresses!

3. Using the scrolling keys, position the old start address in the "active" address line on the screen.
4. Press Key 0 once.

5. Enter three digits of the last address. (See step 1.) Remember not to include the leading zero.
6. Enter three digits of the new start address. (See Step 2.) Again, exclude the leading zero.
7. Press Key 0 again.
8. Adjust target addresses within the shifted block for all GOTO or GOSUB (branch) instructions.

To escape from Relocate Mode prior to step 7, press Key F repeatedly. To escape <u>at</u> step 7, press any key except Key 0.

Zeros will be entered automatically into the vacated addresses.

9. After the data has been relocated, use Key E to enter the forgotten instructions.

Another programming aid that is most useful when used in conjunction with the CHIP-8 Program Editor is the ability to fill all of RAM with "0000" so you don't accidentally "fall" into the routines of a previous program.

The following little machine language program sets the entire VIP memory space to zeros as quick as you can flip the RUN/RESET toggle switch. This routine will eventually write zeros into itself, thus forcing the VIP into an idle state until you flip the toggle switch back down.

Enter this code using the normal ROM monitor - not the CHIP-8 Program Editor. If the editor is present in memory when this routine is executed, the editor will also get "eaten" - as will any program residing in VIP RAM.

```
0000  E1  SEX  R(1) ;Last On-Card Memory Byte
0001  F8  LDI       ;Load "D"
0002  00            ;With zeros
0003  73  STXD      ;Store via X and Decrement X
0004  30  BN        ;Loop until
0005  03            ;Finished
```

Normally such a "trick" would not be acceptable to an expert programmer, but in this case, it's a trick that does the job.

Now load the CHIP-8 Program Editor and you're ready to design a game of your very own!

C H I P - 8 P R O G R A M E D I T O R

CONTROLLING ROUTINES

0000 - 0006	Erase Display Page	
0007 - 0033	Initialization - Video On	
0034 - 0053	Display Addresses & Instructions	
0054 - 0069	Keyboard Scan	
006A - 0071	Scroll Forward	Key F
0072 - 0078	Scroll Backward	Key B
0079 - 007F	Page Forward	Key C
0080 - 008F	Page Backward	Key D
008F - 00B0	Enter Instructions	Key E
00B1 - 00BD	Set up for Key 0	
00BE - 00D5	Address Control For Block Data Move	
00D6 - 00F4	Utility Routines	
00F5 - 00FF	Time Loop	
0100 - 010F	Table For Character Addresses	
0120 - 0126		
0127 - 012D	Shift Right Subroutine	
012E - 0132	Masking Subroutine	
0133 - 013F		
0140 - 01A7	Data Move Routine	

REGISTER ASSIGNMENTS

R0	Display Refresh Pointer
R1	8146 - Interrupt Routine In ROM
R2	Stack Pointer - 01FF
R3	Program Counter
R4	First Display Address - Initially 0200
R5	Used For ROM Character Generator
R6	Display Control
R7	ROM Character Address Table Index
R8	Debouncing And Tones (Used In ROM & Time Loops)
R9	+1 In Interrupt. Not used.
RA	Subroutine To Display Characters
RB	Display Page Pointer
RC	8195 - Keyboard Scan Routine
RD	Subroutine To Shift Right - For Display Purposes
RE	Subroutine To Mask Values - For Display Purposes
RF	Utility Register

PROGRAM LISTING

```
0000  90 51 21 81 3A 00 51 B3
0008  A4 A6 91 B6 BB F8 81 B1
0010  BC B5 F8 01 B2 B7 BA BD
0018  BE F8 46 A1 F8 FF A2 F8
0020  02 B4 F8 95 AC F8 11 AA

0028  F8 28 AD F8 2F AE F8 32
0030  A3 D3 E2 69 94 DE DA 84
0038  DD DA 84 DE DA 16 04 DD
0040  DA 04 DE DA 14 04 DD DA
0048  04 DE DA 86 FC 28 A6 14
0050  FB F0 3A 34 E2 91 AF 2F

0058  22 8F 52 62 E2 E2 3E 57
0060  F8 04 B8 98 3A 63 8F FA
0068  0F 52 F8 0F F3 3A 72 84
0070  30 ED F8 0B F3 3A 79 30
0078  E9 F8 0C F3 3A 80 30 F5

0080  F8 0D F3 3A 8F F8 14 AF
0088  24 2F 8F 3A 88 30 F5 F8
0090  0E F3 3A B1 DC F3 3A B1
0098  DC FE FE FE FE 24 24 54
00A0  DC 04 F4 54 DC FE FE FE

00A8  FE 14 54 DC 04 F4 54 30
00B0  EE DC 3A 54 E2 E2 95 73
00B8  96 73 86 73 97 73 DC B6
00C0  DC FE FE FE FE A6 DC 86
00C8  F4 A6 DC B7 DC FE FE FE

00D0  FE A7 DC 87 F4 A7 F8 01
00D8  BF F8 40 AF DF F8 00 B3
00E0  F8 E4 A3 D3 30 ED 00 00
00E8  00 24 24 24 24 24 24 24
00F0  24 24 24 24 24 F8 10 B8
**
0100  30 39 22 2A 3E 20 24 34
0108  26 28 2E 18 14 1C 10 12
0110  D3 A7 F8 05 AF 07 A5 45
0118  56 86 FC 08 A6 2F 8F 3A
0120  17 86 FF 27 A6 30 10 D3

0128  F6 F6 F6 F6 30 27 D3 FA
0130  0F 30 2E 00 00 00 00 00
0138  00 00 00 00 00 00 00 00
0140  F8 01 B3 F8 47 A3 D3 24
0148  24 94 B5 84 A5 DC 3A A0
```

```
** 00F8  98 3A F8 F8 00 A6 30 34
```

PROGRAM LISTING (Continued)

```
0150  97 52 95 F7 3B 77 95 F3
0158  3A 60 87 52 85 F7 3B 77
0160  16 16 05 57 F8 00 55 15
0168  17 96 52 95 F3 3A 62 86
0170  52 85 F3 3A 62 30 A0 16

0178  85 52 86 F7 AF 95 52 96
0180  77 BF 8F 52 87 F4 A7 9F
0188  52 97 74 B7 25 06 57 F8
0190  00 56 26 27 96 52 95 F3

0198  3A 8D 86 52 85 F3 3A 8D
01A0  60 42 B7 42 A6 42 B6 02
01A8  B5 F8 00 BF F8 DD AF DF
```

C H A R A C T E R D E S I G N E R

C H A R A C T E R D E S I G N E R

The CHIP-8 language was designed primarily to enable simple game and graphics programming on the VIP with minimum memory requirements. That such a miniscule interpreter performs this task so well is a credit to its designers!

However, as programs increase in complexity, it becomes evident that an additional feature is sorely needed: the ability to easily display messages and instructions on the screen. Figuring out all those bit patterns for "YOU WIN", "PICK A CARD", etc., takes too much time and far too many pages of graph paper. Wouldn't it be nice to be able to enter a string (i.e., a series) of ASCII hex codes somewhere in memory and have CHIP-8 do all the work of deciphering the binary patterns to make up the letters? And for keyboard owners: Wouldn't a way to design character sets make life easier than having to calculate all those addresses? How about a full 128-character set for the VIP that can be stored in 500 bytes? Including punctuation, lower-case letters, and math symbols? A tall order, this; but possible!

In searching for a binary answer to these questions (having assumed the program would have to be written in machine language), it occurred to me to ask one more question: Can CHIP-8 be used for things other than games and graphics? Why not program the Character Designer in CHIP-8? Asking questions like these led to my Character Design Program, which uses the 2-Page Display Program described in the September, 1978 issue of the VIPER* - with the minor changes listed later in this chapter.

---

* The VIPER is a newsletter dedicated solely to VIP owners. You can obtain a subscription to it by sending $15.00 to ARESCO (if you live outside the USA, send $25.00). Address is PO Box 1142, Columbia MD 21044.

The Character Designer program is stored from 0300-05FF, and the character set is stored in packed form in 0600-07FF. The program will run in 4K without changes except for the 00EB change described in the 2-Page Display article. The Designer displays a 4 x 8 grid on the screen, and uses a blinking cursor to indicate which square of the grid is to be filled or erased. The cursor is moved with the usual 2, 4, 6, and 8 keys. As you design each character, the resulting pattern is shown to the right of the grid exactly as it will look when you use it. This pattern is referred to as the "sample" character.

Below the sample character, the ASCII code for the character is displayed, along with the address in memory where the bit pattern for the sample characters will be/are stored. Each character takes four bytes of memory. Displaying the entire 128-character set is possible, and a complicated sequence of events must occur before erasure of the character set takes place. In fact, a "warning" note sounds before erasure, to help prevent accidental erasure of the entire character set.

Key C is used as a control key to set up selection of one of several available features of the program. This feature was included to allow for future expansion of the program without disturbing the present contents. If you get confused in the middle of selecting a function, Key F will always return the cursor to the screen. If you are in Mode Two (Display Mode), pressing Key C, followed by Key 1, will get you out of trouble and return you to Mode One (Design Mode).

Changes To The Two-Page Display Program

These changes are to be made to the CHIP-8 Two-Page Display Interpreter found in the September, 1978 issue of VIPER, pages 11 and 12 - <u>in addition to</u> the changes listed by Andy Modla and Jef Winsor:

```
0003  05     For 2K VIP Systems
      09     For 3K VIP Systems
      0D     For 4K VIP Systems

0100  9B FF 01 BB D4
01FA  01 00
0200  13 00
```

0244-0249 are no longer needed

These changes:

1. Locate CHIP-8's variable storage, stack, and work space in 02A0-02FF (V0-VF are at 02F0-02FF now).
2. Put the "Adjust TV Memory Pointer" routine in a more convenient spot.
3. Enable programs to begin at 0300, instead of at 0260, as with the previous version. This is more compatible with VIP's recording scheme, and frees up a large area from 0244-029F for additional CHIP-8 instructions.
4. Allows CHIP-8 programs to go right up to the display page. In other words, if you have 3K in your system, your CHIP-8 program can be stored from 0300 to 09FF.

If you wish, you can disable the initial "Erase Display Pages" instruction by changing 01FC to 30FE.

MODE ONE - CHARACTER DESIGN

The program begins with Mode One. A 4 x 8 grid, an ASCII code, and a 4-digit address are displayed on the screen. If there is a character stored at that address, it will appear to the right of the grid. Its bit pattern will be disassembled into the grid, allowing changes to be made and preventing accidental erasure of a particular character. Keys 2, 4, 6, and 8 move the cursor in the appropriate direction.

The C key must be pressed prior to pressing any of the Mode One (or, for that matter, Mode Two) Command keys. In the list of commands which follows, the C is in parentheses to remind you to press the C key first.

Mode One Commands

Key (C) 0 Clear the contents of the grid and the sample character. The bit pattern for the character stored at the display address is undisturbed.

Key (C) 2 Select Mode Two (Display). See Mode Two Commands for instructions to return to Mode One.

Key (C) A Select a new ASCII code. Enter a two-digit number from 00 to 7F. The bit pattern for that character will be found, disassembled into the grid, and the new ASCII code and address are displayed on the screen. If you try to enter a number higher than 7F, the first digit will automatically be changed to a 7, since 7F is the highest possible ASCII code number.

Key (C) D Deposit current grid contents in the address displayed on the screen, and advance to the next ASCII code. Disassemble the contents into the grid. This feature allows sequential design.

Key (C) E Enter a grid marker at the cursor location and at a corresponding point on the sample character.

Key (C) F Cancel the "Control" selection and return the cursor to the grid.

MODE TWO - DISPLAY

When in Mode One, if Key C is pressed, followed by Key 2, you are automatically moved into Mode Two. The entire character set is displayed on the screen for examination. The display is partitioned into 128 blocks, corresponding to

the 128 ASCII character set (and the 4 bytes each of the 4 x 8 grid).

Mode Two Commands

Key (C) 1 Select Mode One (Design). See Mode One Commands for getting back into Mode Two.

Key (C) E/A Erase the character set from memory. This can be used to clear garbage from the memory space on power-up or when beginning the design of a new set of characters. As soon as you press Key E in this sequence, a warning tone will sound, and the program waits for further instruction. After the warning, press Key A to erase the set and automatically return to Mode One. If any key other than Key A is pressed after the warning note has sounded, you automatically escape the erase sequence and are returned to Mode One.

OPERATION NOTES

Try to use standard ASCII references (41 for "A", etc.), so your program will be compatible with other programs written using the Designer. In later programs in this book, we'll use the characters to display messages in CHIP-8 programs.

Bit patterns for each 4 x 8 grid are stored in memory in packed form, with each grid occupying 4 bytes of memory. In other words, each byte stores two lines of each character.

After designing your characters, record two pages, starting at 0600. This will store the set on tape separately from the designer, and will be used in this form later.

I have included my version of a character set, using 3 x 5

letters and symbols, positioned in the upper right corner of the grid. This allows 16 characters to be displayed on each line, with ten lines and an underline cursor (using the 2-page display). The higher resolution displays, while useful for text applications (as described by Don Stein in issues 3, 4 and 5 of Volume 1 of The VIPER) are not needed here; the goal is to produce English-language statements for CHIP-8 game programs.

I have also included my interpretations of the card suits: spades, diamonds, clubs, and hearts, in that order, using the ASCII codes 1C, 1D, 1E, and 1F respectively.

The Messager Program (which follows shortly) will allow positioning of lines of text anywhere on the screen, at selected X and Y coordinates. "I" is set (using the AMMM instruction) to the desired message (which is stored in ASCII form). The Message Display routine is called, and one standard DXYN instruction displays up to 16 characters!

What used to take hours to draw out on graph paper, and what once required complicated programming, will now take only a few minutes - and what's more, your VIP will do the work!

THE PROGRAM

The code for the program has been listed with comments, so you can follow exactly what's happening. If you purchased the tape containing the program, you should nevertheless try to follow the code and understand the program so you can make your own modifications if you wish to do so.

CHIP-8 VARIABLE ASSIGNMENTS

V0	Key Pressed/Display address. V0 is also used as an escape flag from Control C routine when V0=VF.

V1 - V5	Available
V6	First ASCII digit
V7	Second ASCII digit
V8	Vx for ASCII code & address
V9	Vy for ASCII code & address
VA	Vx for sample character
VB	Vy for sample character
VC	Vx for grid & grid cursor
VD	Vy for grid & grid cursor
VE	Utility variable for various routines
VF	Utility variable for various routines

IMPORTANT PROGRAM ROUTINES

0300	Main program - executive routine
0348	Control C subroutine

CHIP-8 SUBROUTINE ADDRESSES

0368	Cursor blink - Entry #1
036C	- Entry #2
0372	- Entry #3 / Time loop
037E	Mode One - Grid Display Keys Mode Two - C (1)
03B6	Clear Grid Keys - C (0)
03E0	Home cursor. Also resets sample character XY.
03EA	Enter/Erase Grid Mark At Cursor - C (E)
03FA	Accept New ASCII Code - C (A)
043C	Display ASCII Code and Address. 040A entry skips Key-Press part.
0464	Deposit Grid Contents to Memory - C (D)
047E	Mode Two - Display Character Set - C (2)
04DA	Data Storage/Display patterns for DXYN instructions

MACHINE LANGUAGE SUBROUTINES

0500	Unpack Character Bit Pattern
0547	Erase Character Set
0557	Deposit Sample Character in Memory
0581	Display Character Set

0600-07FF	Character Set Storage

The code for the program has been listed with comments, so you can follow exactly what's happening. If you purchased the tape containing the program, don't let that stop you from examining this code so you'll know how it all works.

E X E C U T I V E R O U T I N E S

SEARCH FOR KEY PRESS

```
0300  237E    :Do subroutine - Mode One Grid Reset (Begin Program)
  02  A4FD    :I=Cursor Pattern (Begin Key Search)
  04  DCD1    :Display cursor @ VC, VD (initialized in Mode One
              :subroutine beginning at 037E)
  06  2368    :Do subroutine (cursor blink, entry #1)
  08  6002    :V0=02 (Initialize Keypress variable)
  0A  E0A1    :Skip if V0 ≠ key pressed
  0C  1316    :Go to cursor move routine (Control C check) and
              :end key search.
  0E  7002    :V0+02 (check the next key)
  10  300E    :Skip if V0 = 0E (Last key in sequence)
  12  130A    :GO loop until Key 2, 4, 6, 8, A, or C is pressed.
              :Key A has no function unless it follows Key C.
  14  1306    :GO Loop (Do cursor blink and reset V0)
```

CURSOR MOVE / CONTROL C CHECK

```
0316  236C    :Do subroutine  (cursor blink, entry #2 for erase
              :cursor)
  18  400C    :Skip V0 ≠ 0C (Key C?)
  1A  2348    :Yes: do control C subroutine
  1C  4002    :Skip V0 ≠ 02 (Key 2?)
  1E  4D09    :Yes: limit cursor at top of grid
  20  1326    :Go do next check - not key 2
  22  7DF9    :VD+F9 (-07) - cursor moving up
  24  7BFF    :VB+FF (-01) - sample character pointer moving up
  26  4004    :Skip V0 ≠ 04 (Key 4?)
  28  4C05    :Yes: limit cursor at left of grid
  2A  1330    :Go do next check - not key 4
  2C  7CFB    :VC+FB (-05) - cursor moving left
  2E  7AFF    :VA+FF (-01) - sample character pointer moving left
  30  4006    :Skip V0 ≠ 06 (Key 6?)
  32  4C14    :Yes: limit cursor at right of grid
  34  133A    :Go do next check - not key 6
  36  7C05    :VC+05 - cursor moving right
  38  7A01    :VA+01 - sample character pointer moving right
  3A  4008    :Skip if V0 ≠ 08 (Key 8?)
  3C  4D3A    :Yes: limit cursor at bottom of grid
  3E  1344    :Go exit
  40  7D07    :VD+07 - cursor moving down
  42  7B01    :VB+01 - sample character pointer moving down
  44  2372    :Do subroutine (cursor blink, entry #3 - timing)
  46  1302    :Go loop - end cursor move/control C check
```

END EXECUTIVE ROUTINES

CONTROL "C" SUBROUTINE - CALLED BY KEY "C" PRESS

```
0348  F00A    :V0=hex key.  Wait for instructions
  4A  4000    :Skip V0 ≠ 00 (Key 0?)
```

```
034C  23B6    Yes: Do subroutine (clear grid)
  4E  4002    Skip V0 ≠ 02 (Key 2?)
  50  247E    Yes: Do subroutine (display character set)
  52  400A    Skip V0 ≠ 0A (Key A?)
  54  23FA    Yes: Do subroutine (accept new ASCII code)
  56  400D    Skip V0 ≠ 0D (Key D?)
  58  2464    Yes: Do subroutine (deposit grid contents in
                   memory
  5A  400E    Skip V0 ≠ 0E (Key E?)
  5C  23EA    Yes: Do subroutine (enter/erase at cursor)
  5E  1362    Go to 0362.  No operation.  This space is left
  60  0000         available for expansion (extra key functions)
  62  400F    Skip V0 ≠ 0F (Key F?) or: Escape flag set?
  64  00EE    Yes: Return to executive (Key search/cursor move)
  66  1348    Loop for proper function selection
```

CURSOR BLINK

(Three Entry Points)

```
0368  2372    Do subroutine (time loop for blink) Entry #1
                   (double blink)
  6A  DCD1    Display/Erase cursor
  6C  2372    Do subroutine (time loop for blink) Entry #2
                   (single blink)
  6E  DCD1    Display/Erase cursor
  70  00EE    Return to main program

  72  6F08    VF (utility variable) = 08 (blink timing for
                   entry #3 - time loop)
  74  FF15    Set timer = VF
  76  FF07    Set VF = current timer value
  78  3F00    Skip if VF=00
  7A  1376    Loop until VF=00
  7C  00EE    Return to time loop
```

MODE ONE - GRID DISPLAY

```
037E  0230    Erase Display Pages
  80  A4EC    I=V0-VF (storage array)
  82  FF65    Initialize variables V0-VF (values @ I)
  84  A4FE    I=horizontal line pattern for grid
  86  DCD1    Display @ VC,VD (draw horizontal lines)
  88  7C07    VC+07 (X coordinate)
  8A  3C18    Skip VC=18 (end line)
  8C  1386    Go loop to get to end line
  8E  6C03    VC=03 (reset VC)
  90  7D07    VD=07 (Y coordinate)
  92  3D43    Skip VD=43 (last line)
  94  1386    Go loop to last line
  96  A4FC    I=vertical line pattern for grid
  98  6C03    VC=03 (VX)
```

```
039A  6D04    VD=04 (VY)
  9C  DCD1    Display @ VC, VD (draw vertical lines)
  9E  7D01    VD+01
  A0  3D3D    Skip VD=3D (end  of line)
  A2  139C    Go loop to End of line
  A4  6D04    VD=04 (Reset VD)
  A6  7C05    VC+05
  A8  3C1C    Skip VC=1C (last line)
  AA  139C    Go loop to last line
  AC  A4E2    I=storage array for bit pattern addresses
  AE  F555    V0-V5 stored at I (reset storage array)
  B0  23E0    Do subroutine (home cursor)
  B2  240A    Do subroutine (display ASCII code/address &
                 character
  B4  00EE    Return
```

CLEAR GRID

```
03B6  23E0    Do subroutine (home cursor)
  B8  A4FD    I=cursor pattern
  BA  DCD1    Display @VC, VD
  BC  4F01    Skip VF≠ 01 (no grid marker @ VC, VD)
  BE  23EA    Do subroutine (erase grid marker)
  C0  A4FD    I=cursor pattern
  C2  DCD1    Erase cursor
  C4  7C05    VC+05 (cursor moves right)
  C6  7A01    VA+01 (sample character pointer moves right)
  C8  3C19    Skip if VC=19 (end of row)
* CA  13BA    Loop to end of row
  CE  13DA    Go exit - done.
  D0  6C05    VC=05 (reset cursor X coordinate)
  D2  6A1C    VA=1C (reset sample X coordinate)
  D4  7D07    VD+07 (cursor moves down)
  D6  7B01    VB+01 (sample pointer moves down)
  D8  13BA    Loop until done
  DA  23E0    Do subroutine (home cursor)
  DC  600F    V0=0F (set controls Escape Flag)
  DE  00EE    Return
03CC  403A    Skip if V0 ≠ 3A
```

HOME CURSOR

```
03E0  6A1C    VA=1C (reset cursor X coordinate)
  E2  6B12    VB=12 (reset cursor Y coordinate)
  E4  6C05    VC=05 (reset sample X coordinate)
  E6  6D09    VD=09 (reset sample Y coordinate)
  E8  00EE    Return
```

ENTER/ERASE GRID MARK AT CURSOR - KEY C (E)

```
03EA  A4E8    I=Grid mark
  EC  7DFD    VD+FD (-03) Top of grid square
  EE  DCD4    Display/erase @ VC, VD
```

```
03F0  7D03    VD+03 (reset cursor VY)
  F2  A4FC    I=sample character point
  F4  DAB1    Display @ VA, VB
  F6  600F    V0=0F (set controls Escape Flag)
  F8  00EE    Return
```

ACCEPT NEW ASCII CODE - KEY C (A)

```
03FA  243C    Do subroutine (erase old code and address)
  FC  23B6    Do subroutine (clear grid)
  FE  F60A    V6=key pressed.  Wait for first digit.
0400  6F07    VF=07 (test first ASCII digit)
  02  8F65    VF-V6 (DF into VF)
  04  3F01    Skip VF=01 (VF.GE. V6 - number in 00-07 range)
  06  6607    V6=07 (limits first digit to ASCII 7)
  08  F70A    V7=key pressed.  Wait for second digit.
  0A  0500    Do machine language subroutine (unpack bit pattern
                of character)
  0C  243C    Do subroutine (display new code and address)
  0E  A4DA    I=unpacked character bit pattern
  10  DAB8    Display at VA, VB
  12  A4FC    I=bit for sample character
  14  DAB1    Display on sample character (tests each bit)
  16  3F01    Skip VF=01 (Bit there?)
  18  141E    No: Go display
  1A  23EA    Do subroutine (display grid mark - disassemble
                character into grid)  Bit is there
  1C  1420    Go - skip next instruction
  1E  DAB1    Display bit to erase
  20  7C05    VC+05 (cursor right)
  22  7A01    VA+01 (sample character pointer right)
  24  3C19    Skip VC=19 (end of line)
  26  1412    Loop to test next bit
  28  4D3A    Skip VD ≠ 3A (last line)
  2A  1436    Go exit - done
  2C  6C05    VC=05 (reset cursor X coordinate for next line)
  2E  6A1C    VA=1C (reset sample X coordinate for next line)
  30  7D07    VD+07 (cursor down)
  32  7B01    VB+01 (sample pointer down)
  34  1412    Loop until last line is tested
  36  23E0    Do subroutine (home cursor)
  38  600F    V0=0F (set escape flag)
  3A  00EE    Return
```

DISPLAY ASCII CODE AND ADDRESS

```
043C  F629    I=pattern for V6 (first ASCII digit)
  3E  D895    Display @ V8, V9
  40  7805    V8+05 (for next digit)
  42  F729    I=pattern for V7 (next ASCII digit)
  44  D895    Display @ V8, V9
  46  681C    V8=1C (reset V8)
  48  7907    V9+07 (VY down for address row)
```

```
044A  6E00    VE=00 (Utility loop counter and "I" indexer)
 4C   A4E2    I=storage array for address
 4E   FE1E    I=I+VE (Indexes "I")
 50   F065    Load V0 with byte at "I"
 52   F029    I=pattern for V0 (address digit)
 54   D895    Display @ V8, V9
 56   7805    V8+05 for next digit
 58   7E01    VE+01 (Loop count +01)
 5A   3E04    Skip VE=04 (Done?)
 5C   144C    No:  Loop until done.
 5E   681C    V8=1C (reset X coordinate)
 60   6927    V9=27 (reset Y coordinate)
 62   00EE    Return
```

DEPOSIT GRID CONTENTS IN MEMORY

```
0464  243C    Do subroutine (erase old code)
 66   0557    Do machine language subroutine - deposit sample
               character in memory
 68   23B6    Do subroutine (clear grid)
 6A   7701    V7+01 (ASCII + 01 - for sequential designing)
 6C   3710    Skip V7≠10 (test for carry)
 6E   1478    Go - no carry
 70   6700    V7=00
 72   7601    V6+01 (add carry to V6)
 74   4608    Skip V6=08 (too high)
 76   6600    V6=00 (resets to ASCII 00 after 7F)
 78   240A    Do subroutine (Display ASCII code and character
               at address
 7A   600F    V0=0F (set escape flag)
 7C   00EE    Return
```

DISPLAY CHARACTER SET

```
047E  0230    Do machine language subroutine (erase display pages)
 80   0581    Do machine language subroutine (display character
               set)
 82   F00A    V0=key pressed.  Wait for key
 84   300C    Skip V0=0C
 86   1482    Loop until key "C" is pressed
 88   F00A    V0=key pressed.  Wait for instruction
 8A   4001    Skip V0 ≠ 01  (Key 1?)
 8C   149C    Go exit.  Key 01 resets to Mode One (design)
 8E   300E    Skip V0=0E (first half of erase function)
 90   1488    Loop for proper key sequence
 92   6F70    VF=70
 94   FF18    Sound warning tone for VF
 96   F00A    V0=key pressed.  Wait for instructions
 98   400A    Skip V0 ≠ 0A (Fail-safe from accidental erasure)
 9A   0547    Do machine language subroutine (erase character
               set)
 9C   237E    Do subroutine (reset to Mode One - design)
 9E   600F    V0=0F (set escape flag)
 A0   00EE    Return
```

DATA STORAGE

```
04DA   0000     Unpacked character bit pattern
  DC   0000     Unpacked character bit pattern
  DE   0000     Unpacked character bit pattern
  E0   0000     Unpacked character bit pattern
  E2   0000     Holds address for CHIP-8 subroutines
                   (patterns for DXYN instruction)
  E4   0000     Holds address for CHIP-8 subroutines
  E6   0000     Holds address for machine language subroutines
  E8   C0C0     Grid mark pattern
  EA   C0C0     Grid mark pattern
  EC   0005     V0 V1
  EE   0000     V2 V3
  F0   0500     V4 V5
  F2   0000     V6 V7
  F4   1C27     V8 V9
  F6   1C12     VA VB
  F8   0304     VC VD
  FA   0000     VE VF
  FC   80C0     Line pattern (cursor, sample character point)
  FE   FE00     Line pattern for grid
```

M A C H I N E L A N G U A G E S U B R O U T I N E S

UNPACK CHARACTER BIT PATTERN

```
0500  F8  LDI         ;Load R6 with
  01  F6              ;Address of CHIP-8 V6 variable
  02  A6  PLO   R6    ;R6 points to V6
  03  46  LDA   R6    ;Get V6 value - first ASCII code digit
  04  FE  SHL         ;Shift left, moving
  05  FE  SHL         ;LSB's to MSB position
  06  FE  SHL         ; "        "      "
  07  FE  SHL         ; "        "      "
  08  E2  SEX   R2    ;R2 is Stack Pointer
  09  22  DEC   R2    ;Stack Pointer to free location
  0A  52  STR   R2    ;Push - Stack
  0B  06  LDN   R6    ;Get V7 value - second ASCII code digit
  0C  FA  ANI         ;And "AND" it with
  0D  0F              ;0F for LSB's
  0E  F1  OR          ;Then "OR" with Top of stack - packs ASCII code
  0F  FE  SHL         ;Multiply
0510  FE  SHL         ;by 04
  11  AC  PLO   RC    ;RC indexes bit pattern
  12  F8  LDI         ;
  13  06              ;Base address (high byte) character set
  14  7C  ADCI        ;Add carry, if any, from Multiply Instruction
```

```
0515  00                ;
  16  BC  PHI   RC      ;RC - Indexes bit pattern
  17  F8  LDI           ;Load RD
  18  04                ;With address
  19  BD  PHI   RD      ;Of CHIP-8 storage array
  1A  F8  LDI           ;     "      "       "
  1B  E3                ;     "      "       "
  1C  AD  PLO   RD      ;     "      "       "
  1D  9C  GHI   RC      ;Get address of bit pattern
  1E  5D  STR   RD      ;Store second address digit in array
  1F  1D  INC   RD      ;RD+1
0520  8C  GLO   RC      ;Second half of address
  21  F6  SHR           ;Shift right for third address digit
  22  F6  SHR           ;  "     "          "      "       "
  23  F6  SHR           ;  "     "          "      "       "
  24  F6  SHR           ;  "     "          "      "       "
  25  5D  STR   RD      ;Store in array
  26  1D  INC   RD      ;RD+1
  27  8C  GLO   RC      ;Second half of address - fourth address digit
  28  5D  STR   RD      ;Store in array (MSB's ignored by CHIP-8)
  29  1D  INC   RD      ;RD+1
  2A  9C  GHI   RC      ;Save RC for other machine language subroutines
  2B  5D  STR   RD      ; at 04E2 - 04E3
  2C  1D  INC   RD      ; "   "   "   "       "      "         "
  2D  8C  GLO   RC      ; "   "   "   "       "      "         "
  2E  5D  STR   RD      ; "   "   "   "       "      "         "
  2F  F8  LDI           ;Address bit pattern storage
0530  DA                ;
  31  AD  PLO   RD      ;RD=04DA
  32  F8  LDI           ;Load Utility
  33  04                ;Register with
  34  AF  PLO   RF      ;Loop count
  35  0C  LDN   RC      ;Get Bit pattern @ RC
  36  FA  ANI           ;And "AND" it with
  37  F0                ;F0 for MSB's
  38  5D  STR   RD      ;Store at RD for CHIP-8 DXYN instruction
  39  1D  INC   RD      ;RD+1 - next storage slot
  3A  4C  LDA   RC      ;Same bit pattern - advance pointer
  3B  FE  SHL           ;Shift left for LSB's
  3C  FE  SHL           ;  "     "        "
  3D  FE  SHL           ;  "     "        "
  3E  FE  SHL           ;  "     "        "
  3F  5D  STR   RD      ;Store @ RD for DXYN instruction
0540  1D  INC   RD      ;RD+1 - next storage slot
  41  2F  DEC   RF      ;Loop count (-1)
  42  8F  GLO   RF      ;Test if done
  43  3A  BNE           ;Branch if ≠ 00
  44  35                ;Loop until done to 0535
  45  12  INC   R2      ;Reset Stack Pointer
  46  D4  SEP   R4      ;Return control to CHIP-8
```

ERASE CHARACTER SET - SELECTED KEY IS C (E A)

```
0547  F8  LDI         ;Load RD (Utility
 48   07              ;Register) with
 49   BD  PHI   RD    ;Last address of
 4A   F8  LDI         ;Character Set
 4B   FF              ;    "       "
 4C   AD  PLO   RD    ;    "       "
 4D   ED  SEX   RD    ;X=RD
 4E   F8  LDI         ;Load D with
 4F   00              ;00 - for erasing
0550  73  STXD        ;Store via X (RD) and decrement
 51   9D  GHI   RD    ;Get high byte of RD
 52   FB  XRI         ;Exclusive OR it with
 53   05              ;05 to test if done
 54   3A  BNZ         ;Branch if RD ≠ 05 (page above first
                      ;character set )
 55   4E              ;Branch to 054E - loop until done
 56   D4  SEP   R4    ;Return to CHIP-8 program
```

DEPOSIT SAMPLE CHARACTER IN MEMORY

```
0557  E2  SEX   R2    ;X=R2 (Stack pointer)
 58   22  DEC   R2    ;Point to free location
 59   F8  LDI         ;Load Utility Register RF
 5A   04              ;With loop count of 04
 5B   AF  PLO   RF    ;And also put in RD (RD.1)
 5C   BD  PHI   RD    ;Which will point to the storage
 5D   F8  LDI         ;Area holding ASCII reference
 5E   E6              ;Bit pattern address
 5F   AD  PLO   RD    ; "     "       "    - RD=04E6
0560  4D  LDA   RD    ;Get first part of address and advance pointer
 61   BC  PHI   RC    ;Store in RC.1
 62   0D  LDN   RD    ;Get second half of address
 63   AC  PLO   RC    ;Store in RC.0 (RC is indexed
                      ;to correct address)
 64   9B  GHI   RB    ;Display page
 65   BD  PHI   RD    ;Store in RD.1 (High order sample
                      ;character address)
 66   F8  LDI         ;Load Low order Sample
 67   93              ;Character address
 68   AD  PLO   RD    ;Into RD.0  (RD.0=0Y93)
 69   0D  LDN   RD    ;Get a row of the sample
 6A   FE  SHL         ;Shift left (for packing)
 6B   FE  SHL         ;  "     "    "      "
 6C   FE  SHL         ;  "     "    "      "
 6D   FE  SHL         ;  "     "    "      "
 6E   52  STR   R2    ;Push stack
 6F   8D  GLO   RD    ;Get RD.0 and
0570  FC  ADI         ;Add 08 for next
 71   08              ;row
 72   AD  PLO   RD    ;Put adjusted value back in RD.0
```

```
0573  0D  LDN   RD   ;Get a row of the sample
  74  F1  OR         ;"OR" it with top of stack (packs two
  75  5C  STR   RC   ;sample rows into one byte.  Store in RC)
  76  1C  INC   RC   ;RC+1 - next storage position
  77  8D  GLO   RD   ;Add 8
  78  FC  ADI        ;To RD for
  79  08             ;Another row
  7A  AD  PLO   RD   ;Of sample character.
  7B  2F  DEC   RF   ;Decrement loop count (-1)
  7C  8F  GLO   RF   ;Get RF.0 to
  7D  3A  BNZ        ;Test if done.  Loop until RF=00
  7E  69             ;Branch to 0569
  7F  12  INC   R2   ;Reset Stack Pointer - prepare to return
0580  D4  SEP   R4   ;Return to CHIP-8 control
```

DISPLAY CHARACTER SET - KEY C (D)

```
0581  E2  SEX   R2   ;X=R2 (stack pointer)
  82  22  DEC   R2   ;Point to free location
  83  F8  LDI        ;Load address of
  84  06             ;Character set into
  85  BC  PHI   RC   ;RC
  86  F8  LDI        ;    "       "
  87  00             ;    "       "
  88  AC  PLO   RC   ;    "       " (RC=0600 - base address)
  89  AD  PLO   RD   ;RD=Display page cursor
  8A  9B  GHI   RB   ; "    "     "      "
  8B  BD  PHI   RD   ; "    "     "      "    (RD=0Y00)
  8C  93  GHI   R3   ;R3 is program counter.  Load high address
  8D  BE  PHI   RE   ;Of minor subroutines used by this subroutine
                     ;With RE as program counter.
  8E  F8  LDI        ;Load loop...part one begins....left character
  8F  04             ;Count of four
0590  AF  PLO   RF   ;Into RF.0
  91  0C  LDN   RC   ;Get bit pattern in memory
  92  FA  ANI        ;"AND" it with F0
  93  F0             ;For MSB's
  94  5D  STR   RD   ;Store @ Display page cursor
  95  F8  LDI        ;Load address of
  96  E9             ;Next Line Subroutine
  97  AE  PLO   RE   ;Put it in RE.0
  98  DE  SEP   RE   ;And call the subroutine
  99  4C  LDA   RC   ;Get the same bit pattern (packed)
  9A  FE  SHL        ;Shift Left for LSB's
  9B  FE  SHL        ;(for next row)
  9C  FE  SHL        ;To appear under
  9D  FE  SHL        ;The previous row
  9E  5D  STR   RD   ;Store @ cursor
  9F  DE  SEP   RE   ;Call Next-line Subroutine
05A0  2F  DEC   RF   ;Decrement loop counter (-01)
  A1  8F  GLO   RF   ;Get low RF to test if done
  A2  3A  BNZ        ;Loop to 0591
  A3  91             ;RF=00
```

```
05A4  F8  LDI         ;Load Address of Back Up
  A5  DE              ;Cursor subroutine, and put
  A6  AE  PLO   RE    ;In RE.0
  A7  DE  SEP   RE    ;Then call the subroutine
  A8  F8  LDI         ;Load loop count...Begin part two...
  A9  04              ;Right character...and put the count
  AA  AF  PLO   RF    ;In Utility register RF.0
  AB  0C  LDN   RC    ;Get bit pattern of the character
  AC  F6  SHR         ;Shift right to push MSB's
  AD  F6  SHR         ;To LSB position (right side)
  AE  F6  SHR         ;    "      "
  AF  F6  SHR         ;    "      "
05B0  52  STR   R2    ;Push stack
  B1  0D  LDN   RD    ;Get existing character line on display
  B2  F1  OR          ;page, then "OR" it with Stack
  B3  5D  STR   RD    ;And replace it on display page
  B4  F8  LDI         ;Load address
  B5  E9              ;Of Next-line Subroutine
  B6  AE  PLO   RE    ;Into RE.0
  B7  DE  SEP   RE    ;And call the subroutine
  B8  4C  LDA   RC    ;Get same bit pattern (packed)
  B9  FA  ANI         ;"AND" it with
  BA  0F              ;0F for LSB's
  BB  52  STR   R2    ;Push stack
  BC  0D  LDN   RD    ;Get existing character bit pattern displayed
  BD  F1  OR          ;"OR" it with top of stack
  BE  5D  STR   RD    ;And replace in memory
  BF  DE  SEP   RE    ;Call Next-line Subroutine
05C0  2F  DEC   RF    ;Loop count-01
  C1  8F  GLO   RF    ;To test if done
  C2  3A  BNZ         ;Loop to 05AB
  C3  AB              ;RF=00 (two characters displayed per byte)
  C4  8D  GLO   RD    ;Get current cursor position
  C5  FA  ANI         ;"AND" it with 0F to mask for
  C6  0F              ;LSB's (End of line is always X7)
  C7  FB  XRI         ;Exclusive OR with 07
  C8  07              ;To test if at end of line
  C9  32  BZ          ;Branch if at end of line to
  CA  D7              ;Carriage Return @ 05D7
  CB  F8  LDI         ;Or load address
  CC  DE              ;Of back up cursor subroutine
  CD  AE  PLO   RE    ;Put in RE.0
  CE  DE  SEP   RE    ;And call the subroutine
  CF  1D  INC   RD    ;Next character position (cursor +1)
05D0  9C  GHI   RC    ;Get RC to test
  D1  FB  XRI         ;If at end of
  D2  08              ;Character set
  D3  3A  BNZ         ;If not, loop until
  D4  8E              ;Done to 058E
  D5  12  INC   R2    ;R2+1 (Reset Stack Pointer)
  D6  D4  SEP   R4    ;Return to CHIP-8 control
  D7  8D  GLO   RD    ;Carriage return
  D8  FF  SMI         ;Subtract 07 from cursor
  D9  07              ;To back to beginning of line
```

```
05DA  AD  PLO   RD    ;Put back in RD
  DB  30  BN          ;Branch back, skipping the
  DC  D0              ;Increment RD instruction
```

The following two subroutines are minor subroutines used only in the Display Character Set Subroutine. Use with RE as the program counter.

BACK UP CURSOR

```
05DD  D3  SEP   R3    ;Return leaving RE pointing to entry
  DE  8D  GLO   RD    ;Get current cursor value
  DF  FF  SMI         ;Subtract 40 (hex) to back
05E0  40            ;Up to first character line
  E1  AD  PLO   RD    ;Replace in RD.0
  E2  9D  GHI   RD    ;Get RD.1
  E3  7F  SMBI        ;Subtract borrow, if any, from
  E4  00              ;Previous operation
  E5  BD  PHI   RD    ;Replace in RD.1
  E6  30  BN          ;Branch
  E7  DD              ;To exit @ 05DD
```

NEXT-LINE SUBROUTINE

```
05E8  D3  SEP   R3    ;Return leaving RE pointing to entry
  E9  8D  GLO   RD    ;Get current cursor value
  EA  FC  ADI         ;Add 08 to point to
  EB  08              ;Next character line
  EC  AD  PLO   RD    ;Replace in RD.0
  ED  9D  GHI   RD    ;Get RD.1
  EE  7C  ADCI        ;Add carry, if any, from
  EF  00              ;Previous operation
05F0  BD  PHI   RD    ;Replace in RD.1
  F1  30  BN          ;Branch to exit
  F2  E8              ;At 05E8
```

C H I P - 8 M E S S A G E R

C H I P - 8 M E S S A G E R

The CHIP-8 Messager is a machine language routine that provides the capability of displaying text in programs without having to figure out bit patterns for letters, or calculate each X,Y coordinate for the displays. Sixteen characters may be displayed on a line with any X,Y starting position. Text is stored in memory in standard ASCII encoded form, with all character strings ending with a null (00) character.

The Messager requires a character set - which will provide the bit patterns for each character - with the patterns packed in four bytes each; two rows of four bits for each character per byte. This is the form generated by the Character Designer program discussed earlier in the book. Any changes to be made to the character set, or the addition of graphics, are easy to make by using that program. All the capital letters and punctuation are available for your own programs, starting with a copy of the 2-page interpreter supplied with Character Designer. Enter the Messager program, record three pages, beginning at 0000, and you're ready to begin.

After entering the code for Messager, you need to enter:

1. At 025E - the page where the character set is located.
2. At 024D and 028B - the code for whichever CHIP-8 variables you choose to use for X and Y coordinates. To do this, set the values to FN, where N is the first of the variable pair. For example, to use V5 and V6 for X and Y, respectively, enter F5 into locations 024D and 028B. To use V0 and V1, enter F0 into these locations, and to use VA and VB, enter FA. The variables are always available for other uses in your program.

With a little planning, you can avoid having to make these changes for every new game or message you employ in games. Use a "standard" location for the storage of your character set (I've used 0600 -), and use a "standard" pair of variables for X and Y coordinates (I use V5 and V6).

To display a message, the following instruction sequence must be employed:

6XKK - Select the X coordinate for the display
6YKK - Select the Y coordinate for the display
AMMM - Set I to the memory location of the character set.
0244 - Call the Messager routine
DXYN - Display the message. N may be up to 8 bytes deep, but 5 is enough for most displays.

Up to sixteen characters will then be displayed at your selected XY coordinates. The routine "knows" the line is finished when it "sees" the null character (00) at the end of your message. It will continue to display characters, wrapping around from side to side, until a null is encountered. This will permit simple programming of a "Times Square" type of message sign. Repeating the above sequence will erase the message, just as the normal repeating of the DXYN instruction does. Of course, the DXYN instruction may be used in the normal way, too. Only the 0244 call to the Messager routine <u>immediately</u> prior to DXYN modifies this instruction for displaying messages.

After each use of the Messager routine, the following conditions exist:

1. The selected X and Y coordinates are unchanged.
2. "I" points to the byte immediately following the null character at the end of your message. This feature allows fast display of several lines of text with only one AMMM instruction.

Adding messages to your programs will enhance your graphics and make displaying instructions, player names, etc., much easier than before. It is possible you will want to interface a keyboard to use with this feature, although it was designed primarily to allow simple message display in CHIP-8 programs without any hardware modifications.

In addition to the changes already incorporated into the Character Designer Program presented earlier, you should enter the following alterations to the Two-Page Display Program from the September '78 VIPER:

0211 0F This change allows any page to be the first (top half) on display. As presented in VIPER, the interrupt routine only allows even-numbered pages to begin the display. This modification will in no way affect program you may already have written, but it is necessary in order for the Space Wars Program (coming up next) to run on 4K VIP systems.

01F2 F8 00 B4 F8 1B A4 12 D4 This change provides the capability for resetting R4 for a return (D4) instruction after the Messager is finished with a run. It can be left in place for use with other subroutines which use R4 as a program counter. The stack pointer here is incremented with the instruction at 01F8.

The program listing on the following pages is commented so you can follow the code and understand how the Messager works. If you have purchased the tape with the program on it, you will still learn a lot from understanding the code, and will be able to see how to modify it if you want to do so.

SUMMARY

To use the Messager, a character set must have been previously generated and stored in the format described. The CHIP-8 sequence: 6XKK, 6YKK, AMMM, 0244, DXYN must be employed to display the message.

C H I P - 8 M E S S A G E R

PROGRAM LISTING

```
0244  22  DEC   R2  ;Stack to free location
0245  15  INC   R5  ;Point to second half of DXYN instruction
0246  93  GHI   R3  ;R3=PC here
0247  B4  PHI   R4  ;
0248  F8  LDI       ;
0249  4C            ;Prepare R4 to become PC
024A  A4  PLO   R4  ;
024B  D4  SEP   R4  ;R4 becomes PC
024C  F8  LDI       ;
024D  F5            ;
024E  A6  PLO   R6  ;R6 points to CHIP-8 variable VX.  R6.1
024F  A7  PLO   R7  ;Was set in Fetch routine.  R7 is VY
0250  17  INC   R7  ;
0251  06  LDN   R6  ;Get value of VX
0252  BF  PHI   RF  ;And save it in RF.1 for later reset
0253  9A  GHI   RA  ;
0254  73  STXD      ;Push RA.1
0255  8A  GLO   RA  ;
0256  52            ;Push RA.0 - saves ASCII code pointer
0257  4A  LDA   RA  ;Get ASCII code - advance pointer
0258  32  BZ        ;Branch if null (00) - end of line
0259  95            ;To Exit routine
025A  FE  SHL       ;Multiply
025B  FE  SHL       ;By 04
025C  AC  PLO   RC  ;
025D  F8  LDI       ;
025E  07            ;Page address of character set
025F  7C  ADCI      ;Add carry, if any, from
0260  00            ;The Multiply instruction
0261  BC  PHI   RC  ;RC is indexed to the character bit pattern
0262  1C  INC   RC  ;Points to last bit pattern
0263  1C  INC   RC  ;  "     "   "    "     "
0264  1C  INC   RC  ;  "     "   "    "     "
0265  94  GHI   R4  ;(=02)
0266  BA  PHI   RA  ;
0267  F8  LDI       ;
0268  EF            ;
0269  AA  PLO   RA  ;RA points to CHIP-8 work area
026A  F8  LDI       ;
026B  04            ;
026C  AF  PLO   RF  ;Loop count
026D  2A  DEC   RA  ;
026E  0C  LDN   RC  ;Get pattern
026F  FE  SHL       ;Shift left for LSB's
```

```
0270  FE  SHL       ;Shift left for LSB's
0271  FE  SHL       ;  "     "    "    "
0272  FE  SHL       ;  "     "    "    "
0273  5A  STR   RA  ;
0274  2A  DEC   RA  ;
0275  0C  LDN   RC  ;
0276  FA  ANI       ;And with F0
0277  F0            ;For MSB's
0278  5A  STR   RA  ;
0279  2C  DEC   RC  ;Next bits
027A  2F  DEC   RF  ;Loop count-1
027B  8F  GLO   RF  ;
027C  3A  BNZ       ;
027D  6D            ;Loop until done - bit pattern unpacked @ M(R(A))
027E  B3  PHI   R3  ;(00-R3.1)
027F  F8  LDI       ;
0280  70            ;Address DXYN routine
0281  A3  PLO   R3  ;R3 is PC for Display routine
0282  D3  SEP   R3  ;Do display routine
0283  E2  SEX   R2  ;Reset stack pointer to 2 on return
0284  25  DEC   R5  ;Point back to second half of DXYN Instruction
0285  72  LDXA      ;Pop the saved RA.0
0286  AA  PLO   RA  ;
0287  F0  LDX       ;Pop the saved RA.1
0288  BA  PHI   RA  ;Restore RA value
0289  1A  INC   RA  ;RA points to next ASCII code
028A  F8  LDI       ;
028B  F5            ;
028C  A6  PLO   R6  ;R6 points to VX again
028D  A7  PLO   R7  ;
028E  17  INC   R7  ;R7 points to VY again (or R6 + 1)
028F  06  LDN   R6  ;Get VX value
0290  FC  ADI       ;Add 04
0291  04            ;
0292  56  STR   R6  ;And store via R6
0293  30  BN        ;Loop until done - all
0294  53            ;Characters displayed
0295  15  INC   R5  ;Point to next CHIP-8 instruction
0296  9F  GHI   RF  ;Saved value of VX
0297  56  STR   R6  ;Replace in CHIP-8 variable storage area
0298  12  INC   R2  ;To counter the last STXD instruction
0299  F8  LDI       ;Load Address
029A  01            ;
029B  B3  PHI   R3  ;Of return sub @ 01F2
029C  F8  LDI       ;
029D  F2            ;Into R3
029E  A3  PLO   R3  ;And call to begin
029F  D3  SEP   R3  ;Return
```

S P A C E W A R S

S P A C E W A R S

INSTRUCTIONS

You are the captain of the Starship COSMAC - on a mission through dangerous enemy territory. Klingon war vessels attack! You're stronger than they are, but your shields will withstand only four hits. Your phaser banks are set for 30 blasts, and your object is to shoot down as many Klingon cruisers as you can - and make it safely through their sectors of space. If you're hit four times, your starship will be destroyed, your crew lost - and worse, you will have failed in accomplishing your mission.

You must score a direct hit at the apex of your double phasers to destroy an enemy target. Either or both of the last bips of the phaser stream, if they touch any part of the target, will explode it. When hit, the target shudders momentarily, as if trying to absorb the impact of your phaser blast; then it is destroyed in an explosion that sends debris flying off into space.

When your starship is hit, you'll see the flashes of the Klingon rays pounding at your forcefields. Your controls will be temporarily disabled while all available power is diverted to your shields, and you can't fire back.

Key "F" fires your phasers, and the starship responds to your instructions with keys 2, 4, 6, or 8 to bring the enemy tar- into range. Four "cross hairs" help you sight the enemy craft. Because you are looking toward the Klingon fighters, you must move toward the target to bring it into range. Pressing Key 2 will bring the target down; Key 8 brings it up as your sights move down to it. This may take some getting used to - but at least you don't have to feed your VIP a quarter for each new game!

At the end of the game - which comes when you run out of phasers or are shot down - you'll learn if you made it through

the enemy lines, how many enemy ships you hit and whether you are reading a post-mortem (your ship has been destroyed). Instructions are also displayed for starting a new game (Press Key F).

Because your shields come on automatically when you're being fired at, you can't return fire at all during that time. This is true, too, for the Klingons. In addition, your starship has inertia (as would be expected in deep space), and will continue to rotate in the direction selected until you change it. After playing a few times, you'll notice that the direction keys don't have to be held down continuously. Press them once for rotation to begin. This gives the keypad a light touch. However, when you fire, your sights are "locked in", although the target is free to take evasive action and will continue to move.

These features allow for very realistic movement. The target and your starship appear to float in space separately from each other, and it will probably take you some time to master the controls. You must maneuver fast - or you risk being destroyed!

The targets aren't easy to hit. They don't hold still for you (but who would expect that of a Klingon warrior?), so if you manage to hit three (10% of your firing power), you've done well. Five or more is very good indeed, and if you hit ten, you're an excellent shot! This is a very realistic battle simulation. While developing the game, my wife, friends, and I played several hundred rounds, and I currently hold the record at only nine Klingons! Scores of zero are frequent and add to the challenge.

Good luck on your voyage!

S P A C E W A R S

Space Wars is a battle simulation game that uses a modified, Two-Page CHIP-8 Interpreter that is loaded into memory locations 0000-02FF. The modifications are detailed in the section titled "CHIP-8 Messager", and do not affect the normal operation of CHIP-8 programs except that 0300 is now the normal start address for programs. The CHIP-8 stack and work spaces are located on page 3 (0200) - a more convenient spot.

Space Wars is provided with a modified character set of 128 characters. If you examine the set with the Character Designer program, you will see the addition of the title in place of several of the lower case letters.

Once the modified interpreter is loaded, two changes need to be made (if they are not part of your "standard" usage):

1) At location 025E, enter 07. This is the page where the character set is stored.
2) At locations 024D and 028B, enter F5. These are the addresses containing the code for the first of the XY coordinate pairs used for the display.

Please refer frequently throughout this discussion to the flowcharts and program listing. This section of the chapter describes how the program was written and gives hints on modifying certain parameters for a "customized" game. If you decide to make any changes, however, it is strongly recommended that you preserve the original game on tape in case a bug develops from the modifications. (This is the "curse of the programmer", and occurs most frequently when more than 1 instruction is modified in any given program.)

As some of the routines used by Space Wars are discussed in an earlier program (See "Surround"), they are not detailed again here. However, the sections affected are indicated in case you want to look them up.

Space Wars contains many features that could be useful for a variety of games, and you are encouraged to "tinker" with the program to discover how it works. I will try to concentrate on some of the more interesting details here.

The main program loop exists at locations 0300-038C. Actually, the first part simply sets the stage, draws the phaser sights, and calls the subroutines to do the work. The action begins at location 0332. The remainder of the program consists only of subroutine modules which are called by the main loop. Some of the subroutines call others, of course - a technique known as "nesting". Twelve levels of nesting are available with CHIP-8 - which means you could have twelve subroutines, each of which is called by a previous one before control is returned to the main program. Such depth is probably very rare, and is usually not necessary.

I strongly recommend this "modular" form of programming, by the way. The idea is to write down all the functions your program will require, concentrating on what each section is to do, rather than on how you will program it to perform. For instance, in Star Wars, a partial list includes such things as: Fire Phasers, Display A Target; Return Fire; etc. These function names later became the titles for the subroutines as each routine was programmed and debugged.

The next step is to write the main loop, or executive routine. This routine initializes everything of importance and calls the subroutines in their proper sequence. This loop should be capable of functioning without any of the subroutines it will eventually call (as each subroutine should be able to function alone to perform its function). Then, as each routine is written, it can be tested and debugged without

affecting previously programmed sections. Space Wars was programmed in exactly this manner. The routines together required about ten hours to write and debug (Of course, I spent twice that much time - at least - playing Dr. Frankenstein. My Star ship was continually being obliterated by the monster I had created!)

The first instruction, at 0300, passes control to the message routines for displaying the title. Use of the CHIP-8 Messager will enhance your game programming, and it is utilized liberally in Space Wars as well.

In locations 0302-0304, all the variables from the data array (at location 05F0) are initialized. The phaser sight are constructed in 0306-032E using the "brute force" (as opposed to the "clever") method. I prefer to get the "housekeeping" chores done and out of the way first, leaving more time for the parts of the programming that really matter. Use of a data array will be discussed in "Surround", and you may want to peek ahead and look at that section.

Locations 0332-0352 are practically identical to the use in many VIP games of Keys 2, 4, 6, and 8. The difference here is in the effect of each key.

In the VIP Manual's game TANK, for example, when you press a key, you indicate the direction in which you want to move a pattern on the screen. Here, in Star Wars, I wanted the player to have the effect of looking through a phaser sight, so that to bring the target into range, it is necessary to press the key in the direction you want to move to accomplish that objective, rather than pressing a key which would move the target to where you wanted it to be.

Obviously, the phaser sights do not actually move. Only the target does. And to give the appearance of phaser sight

movement, the opposite values (opposite to those in Tank or Surround) are added to the target. Thus, for a key 2 press, the target's Y coordinate is increased by one, and the phaser sights appear to raise. Actually, the target is moving down. Because the target itself is continually being supplied with new and random X,Y coordinates at the same time, both the starship's and the target's movements have the appearance of being separate and distinct. They are, in reality, one and the same. I am very pleased with the realistic effects of this feature, and hope you'll try it for your own "battle" programs. A joystick would be a great addition, such as the one described in an early VIPER.

Line 0354 calls the first subroutine of the main loop. We'll discuss all the subroutines in order after we've finished with the executive routine here.

Locations 0356-035C do a quick check to see if the "fire-button" (Key F) has been pressed. (Left-handers may want to change the fire button to some other key by setting a new value into V0 at location 0356.) If the "fire button" has been pressed, the Fire Phasers Subroutine is called, a tone is sounded, and the phaser "banks" are depleted by one phaser. Upon return from this subroutine, either V0 or V1 equalling 1 indicates a "hit", and to test this condition, VF is added to V0 (for simplicity, since now only one test will be needed), and the value of V0 is checked. If V0 is not zero, then there was a hit. The number-of-hits counter, VD, is increased by one, the target is erased, and the Enemy Craft Destruct subroutine is called.

As an interesting aside: To illustrate the programming tech - nique outlined above, try taking out the subroutine call at 0372, replacing it with a 1374 (GOTO 0374) instruction. The game will function exactly as before, except that now, instead of the target exploding, it will simply disappear when hit. The destruct sequence could have been written and inserted

at any time, since it doesn't provide any data to the rest of the program. You could, for example, decide to write a new destruct routine, with a more elaborate explosion sequence which throws all sorts of garbage out into deep space each time you score a hit. And you simply replace the subroutine call with a call to your own subroutine to complete the "interfacing" between this existing program and your enhancement.

To continue: At location 0374, you will notice a flag - V1 - being set equal to 1. This was previously set to zero, and provides a signal to the phaser firing routine that a target was hit and no longer exists on the screen. As the target coordinates are updated between each of the phaser "bips", the routine must know if there is a target on the screen or not. The target continues to move while firing. Then, when the phaser fire routine is repeated by the subroutine call at 0376 in order to erase the phaser lines, the sub will move the target around or not, as needed.

Lines 0378-037E use a technique described in issue 1, Volume 1 of The VIPER, on making random decisions as to program flow. For your information, I have included the percentages to show the frequency with which the target will "decide" to return fire. This frequency may be changed with a new "KK" mask at 0378, but I don't know why you'd want to have it firing more often. It hits me all the time!

Next, either the target is erased or the instruction ignored (depending on the value of V1). End-game conditions are checked, and if the conditions are not met, the game continues with a jump back to 0332. If you feel your starship is being destroyed too quickly, change the value of the skip condition at 0384 to a higher number. Or live dangerously and choose a lower value.

The Fire Phasers Subroutine at 038E is a very simple display of bips angled toward center screen. The phaser length can not be easily changed, as the two sides must not run into

each other and register false hits. V1 is the flag discussed above which enables or disables target movement while firing. (This happens with the subroutine call at 039E.) V0 and VF are used to indicate hits, allowing either of the last bips of the phaser to score. As only the end result of V0 or VF counts, the target must be hit at the exact apex of the phaser for a hit to be scored. Because the target can move over the phaser trail after it is displayed, it occasionally may appear that a target was hit when it in fact was not. (I tell my friends, when they come over to play Space Wars, that the target was out of range, so its "force field" absorbed the phaser fire. They say, "Really? How'd you program that?")

One other feature of the firing routine is the cancellation of the starship inertia in order to steady the target. Remember - this is only the value of your 2, 4, 6, or 8 Key press, added to the XY coordinate values. This is accomplished by a second entry point into the Target Move subroutine that skips the instructions adding the phaser sight-movement values. If this were not done, the target would continue to travel in the direction you selected while the phaser was firing. I discovered that the only way to hit a target with the inertia enabled was to lead it by half a screen! (You can try this for yourself by changing the call at 039E to 23B6.) I describe this as the "phaser lock in" feature for the game instructions. The target will stay in the same area when you press the fire button, but will continue to move randomly within that area. This gives the target the ability to take "evasive action". But not too much!

The Display New Target subroutine at 03B6 adds the phaser sight adjusters to the target XY coordinates, then randomly chooses new XY directions so that the target appears to move independently. The target must be off when entering the

routine, and a new target will be on when exiting. This routine in turn calls the next subroutine, at locations 03D4-03E4, which cancels the wrap-around effect of the target. This effect made the movement of the target difficult to control, and detracted from the realism of the game. If you want to see whether or not you agree, enter an 00EE at location 03D4 (Return from subroutine instruction). You can also try a version which eliminates the calls to this routine in the Display New Target Subroutine. Or perhaps you'll want to include a feature in which the player can select wrap-around or not. What would be a good way to permit this? Do I hear someone suggesting self-modifying programs??? Good Heavens!

The target destruct sequence at 03E6 blows the target up after it has been hit. Each pattern of destruction was first drawn and then stored in the data section (beginning in 05C6). The patterns are displayed in order by repeated calls to the next subroutine, simply changing the "I" pointer for each pattern. The Display Debris subroutine at 0418 also calls a timer subroutine (described later in the program details given for Surround). Thus we have subroutine nesting here to a depth of three.

The section at 0406 randomly positions a new target in one of the four corners of the display, and at 0422 the Target Returns Fire subroutine begins. This routine, which lasts for less than half a second, produces an effect I really like. When you're running the program, look closely at the phaser fire coming from the target. Notice the transparency. This 3-D effect is easily achieved and adds realism to the graphics. The target's phaser is really a series of three blocks, each larger than the last. These are displayed in a cycle of four, and each block is erased immediately after it is displayed. This gives the appearance of the phaser coming toward you (any object appears to increase in size as it grows closer). To give the illusion of transparency, the target is erased

and then displayed, alternately between each cycle. The eye is fooled into seeing a stable target all the time, with the phaser fire emanating from it. It also gives the phaser its quick, pulsating effect. The target is erased first so the routine returns with the target on, though other variations are possible.

Looking closely at the routine, you'll see the redundancy of the display and XY coordinate adjustments. I wrote four versions of this routine, searching for a more efficient loop, but was only able to cut two lines (including the bytes for the patterns) by converting the redundancies to a subroutine. Since the three subroutine calls (and three returns) added more time to the loop than I saved by cutting two lines, the redundant code stayed. Losing the fast pulse of the target's phaser fire would detract from the game, I felt. I could have displayed and erased the ship more often for less of the "blinking" effect, but again, the speed would suffer. In this version, the target stays on quite long enough to fool the eye, but for more complex transparent graphics, you'll have to have many more DXYN instructions to get the desired effect.

Next, and in the same routine, the program uses the same random decision technique to decide if your starship has been hit. Though generating a random number from 00-07 will produce zero 12.5% of the time, the routine itself is called randomly. Therefore, your starship is hit only a percentage (12.5%) of the time that the program decides to allow the target to fire (3.12% of the time). So your chances of being hit are really only 0.25% (3.125 ÷ 12.5) of the times the program passes through the main loop. This is a delicate balance which seems to produce very competitive results. (That means the target hits you about the same number of times you're able to hit it.) By all means, play around with some

different CXKK masks, but a little change here will produce drastic effects. An easier way to change the competitiveness of the game is to adjust the number of times the starship gets hit before it is destroyed.

The technique of nesting random decision points may be used to weight a complex decision "tree" in which many decisions are affected by the decisions made elsewhere. This is helpful in game simulation type programs. An excellent game of this type is "The Oregon Trail", which appeared in an issue of Creative Computing. The game is based on actual diaries and recorded frequencies of events, and demonstrates how such real situations can be translated into a decision tree. You need a BASIC-speaking computer to run that particular game, but the same concept can be applied to your VIP programs.

The remaining two CHIP-8 subroutines simply display the score and provide a timer for graphics and for the message routines.

At 054E, we wait for an instruction to start a new game - by pressing Key F. The display is simply erased and a jump is performed - back to the beginning at 0300. Please note the new erase instruction, 0230, which replaces the 00E0 instruction used for one-page CHIP-8 programs.

The only machine language subroutine in Space Wars is used to flash the screen on and off to show when the starship has been hit. It was written at location 0558 in machine language strictly for speed, and simply flips the display pages back and forth to an area completely filled with hexadecimal FF's. The reason for the two different routines - one for 3K and one for 4K systems - is one of space. In a 3K system, there isn't enough room to store two pages of ones. Playing a hunch, I tried switching to non-existent RAM locations directly above the display. The VIP sees all logic 1's at this point (I assume it's because of ground connections), and we get two pages of white that aren't really there!

Unfortunately, this can't be done in 4K systems, and the two pages must be loaded from tape. Timing loops within this subroutine keep the flashing at a reasonable rate.

Other changes you can make include:

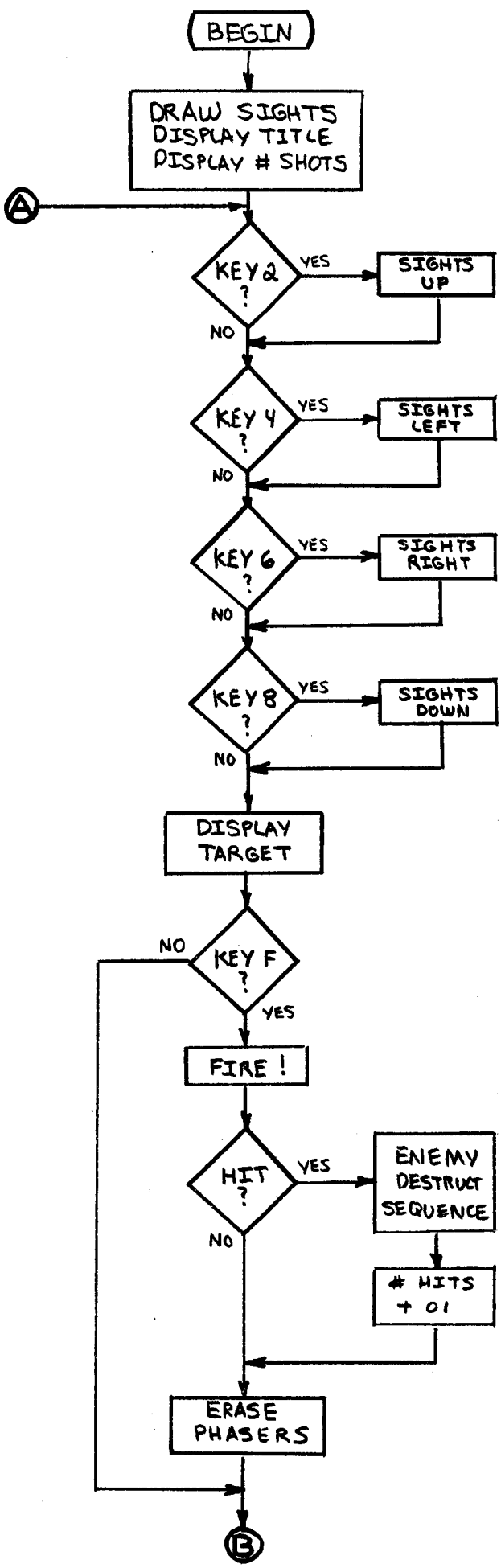
1) My VIP Manual states that 01 will not work as a value for the tone generator. You should hear a "beep" every time the target is hit. If you don't, try changing VE to a value of 02 at location 03E6.

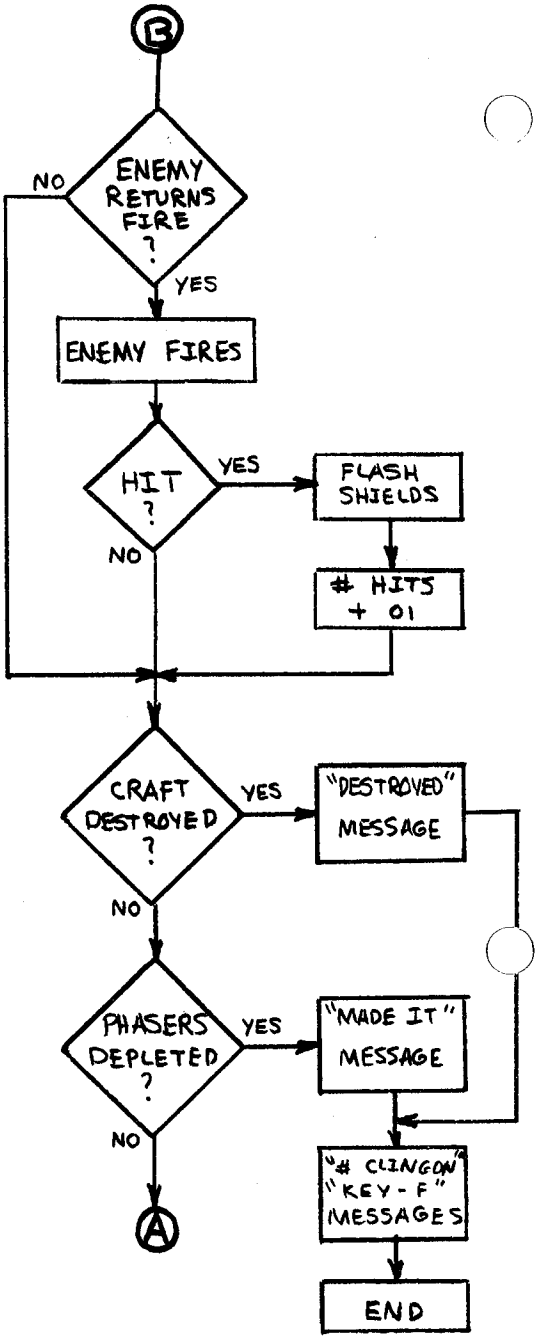
2) The target pattern is stored in the data section, at location 05EA. You have four bytes in which to draw different ships if you wish.

3) The initial value of VC (the number of Phasers) is in location 05FC. Changing this value to a higher or lower value alters the number of shots. However, you'll also want to change the message at the beginning of the game which states how many phasers are available. This is stored at 062C, in ASCII-encoded form. Change these two bytes to 3X 3Y where XY equals the two digit number of phasers. All ASCII numbers begin with a hex 3 in the first digit position.

4) See instructions on the use of the CHIP-8 Messager to enter your own messages for a custom game. You might experiment with changing the ASCII codes at 0600, but you're limited to the same number of characters as are already there. All character strings must end with a null character (00).

If you have troubles, use the checksum data for both the game and for the modified interpreter. I hope you enjoy playing Space Wars, and will try some of the modifications suggested to learn more about your VIP.





SPACE WARS	
COSMAC VIP	
T. SWAN	1979

S P A C E W A R S

Variable Assignment

V0 Used in key scans. Also shows hit with VF after firing.
V1 Target-movement flag. Enables target movement while firing
V2 Used with V1 to display score
V3 X Coordinate adjuster for phaser sight movement
V4 Y Coordinate adjuster for phaser sight movement
V5
V6 XY coordinate and loop counts for phasor firing and for
V7 message displays
V8
V9 X Coordinate for target (also for destruct and return-fire)
VA Y Coordinate for target (also for destruct and return-fire)
VB Number of times starship is hit: initially 0; at 4, destroys your starship
VC Number of phasers. 0 ends game. Initially 1E (30 shots)
VD Number of times the target is hit. 3 is average, 6 is very good, 10 is excellent. Initially is 0.
VE Utility loops, miscellaneous flags, loop counts. Passes
VF time value to timer subroutine. VF is also hit indicator

Program Structure - All Systems

0000-02FF Modified CHIP-8 Two-Page Interpreter with "Messager"
0300-038C Main program; initialization, key presses, scoring
038E-03B4 Fire phasers
03B6-03D2 Display new target. Entry at 03BC disables phaser sight movement.
03D4-03E4 Limit target to screen edges
03E6-0416 Target Destruct display
0418-0420 Display debris
0422-045A Target returns fire. Also decides if starship was hit.
045C-0470 Display the score
0472-047A Timer. VE passes value.

Other Routines And Data - 3K & 4K Systems

047C-0556 Message routines
0558-0574 Machine language subroutine - starship hit sequence (4K system owners note changes)
0576-05C0
05C2-05FE Data storage. Patterns for display, etc.
0600-06A2 ASCII Coded messages
06A4-06FE
0700-08FF Character set - modified 128-character set.
0900-0AFF All FF's for 4K system only. Used by MLS starship hit sequence
0Y00-0XFF Top two memory pages for display - all systems.

S P A C E W A R S

PROGRAM LISTING

ADDRESS	CODE	COMMENTS
0300	147C	Go to Message Routine for title
02	A5F0	I= storage array (CHIP-8 variables)
04	FF65	V0-VF = data at I: Initialize the variables
06	A5EE	I= "bip" to draw phaser sights
08	D011	Display @ V0, V1
0A	7102	V1=V1+02 (VY)
0C	310E	Skip if V1=0E
0E	1308	Loop until done
0310	613F	V1=3F; next line
12	D011	Display @ V0, V1
14	71FE	V1=V1+FE (-2)
16	3131	Skip if V1=31
18	1312	Loop until done
1A	6000	V0=0
1C	611F	V1=1F
1E	D011	Display @ V0, V1
0320	7002	V0=V0+02 (VX)
22	300A	Skip if V0=0A
24	131E	Loop until done
26	603F	V0=3F
28	D011	Display @ V0, V1
2A	70FE	V0=V0+FE (VX - 02)
2C	3035	Skip if V0=35
2E	1328	Loop until done

CHECK IF KEYS 2, 4, 6, or 8 HAVE BEEN PRESSED

0330		14AC	Go to Message routine for game beginning
32		6000	V0=0
34		7002	V0=V0+2 (cycles through 2, 4, 6, and 8)
36		E0A1	Skip if V0 ≠ key pressed
38		1340	Go adjust starship movement - a key was pressed
3A		400A	Skip if V0 ≠ 0A: Done with one cycle?
3C	*	1344	Go - continue in direction last selected.
3E		1334	Loop until done or key 2, 4, 6, or 8 pressed

* NOTE: Line 033C could jump ahead even further, but it cycles through the next part for timing realism.

```
0340    6300    V3=0: Reset movement adjusters
  42    6400    V4=0:   "       "       "
  44    4002    Skip if V0 ≠ 2 (Key 2 not pressed)
  46    6401    V4 = 1 : Sights up (target moves down)
  48    4004    Skip if V0 ≠ 4 (Key 4 not pressed)
  4A    6301    V3 = 1 : Sights left (target moves right)
  4C    4006    Skip if V0 ≠ 6 (Key 6 not pressed)
  4E    63FF    V3=FF (-1): Sights right (target moves left)

0350    4008    Skip if V0 ≠ 8 (Key 8 not pressed)
  52    64FF    V4=FF (-1): Sights down (target moves up)
  54    23B6    Do subroutine - display target at adjusted
                coordinates
```

FIRE PHASERS - SCORING - TARGET RETURNS FIRE

```
0356    600F    V0=0F (Key "F" check)
  58    6100    V1=0: Set target-move flag (allows target
                to move while you are firing)
  5A    E09E    Skip if V0 = key pressed ("FIRE!")
  5C    1378    Go - no fire
  5E    6E28    VE=28 for phaser tone

0360    FE18    Sound tone for VE
  62    7CFF    VC=VC+FF (phaser banks -1 shot)
  64    238E    Do subroutine Fire Phasers.  V0 or VF =1
                indicates a hit
  66    80F4    V0+VF : Add the two hit indicators
  68    4000    Skip if V0 ≠ 0 (skips when hit indicated)
  6A    1376    No hit, so skip next few instructions
  6C    7D01    VD=VD+1 (Number of hits +1)
  6E    A5EA    I=target pattern

0370    D9A4    Display to erase @ V9, VA
  72    23E6    Do subroutine (enemy craft destruct sequence)
  74    6101    V1=0 : disable target-move flag for phaser
                move sequence
  76    238E    Do subroutine - erase phasers
  78    CE1F    VE=RND (random number between 0 and1F)
  7A    4E00    Skip if VE ≠ 0 (96.875% of the time)
  7C    2422    Do subroutine - target returns fire (3.125%
                of the time)
  7E    A5EA    I=target pattern

0380    3101    Skip if V1=1 : target move disabled, no target
                on the screen
  82    D9A4    Display to erase @ V9, VA
  84    4B04    Skip if VB≠4 (maximum number of hits on starship)
  86    14E2    Go to end : "Your starship is destroyed"
  88    4C00    Skip if VC ≠ 0 (Phaser banks depleted?)
  8A    1500    Go to end : "You made it"
  8C    1332    Continue

        *       END OF MAIN PROGRAM LOOP    *
```

S U B R O U T I N E S

FIRE PHASERS

```
038E    663F    V6=3F : Y Coordinate
  90    6700    V7=0  : X Coordinate #1
  92    683F    V8=3F : X Coordinate #2
  94    6510    V5=10 : Loop counter for phaser length
  96    3100    Skip if V1=0 (move target flag set by calling routine)
  98    13A0    Go - target off - skip next instructions
  9A    A5EA    I=target pattern
  9C    D9A4    Display to erase old target
  9E    23BC    Do subroutine to display new target without
                phaser sight inertia
03A0    A5EE    I=phaser bip for display
  A2    D761    Display one side phaser @ V7, V6
  A4    80F0    V0=VF : saves last value of VF (hit indicator)
                in V0 (VF indicates hit)
  A6    D861    Display other side phaser @ V8, V6
  A8    76FE    V6=V6+FE (Y coordinate -2) ---+ Angles phasers
  AA    78FE    V8=V8+FE (X coord. #2 -2)     | inwards toward
  AC    7702    V7=V7+2  (X coord. #1 +2)  ---+ screen center
  AE    75FF    V5=V5+FF (loop count -1)

03B0    3500    Skip if V5=0 : Done yet?
  B2    1396    Loop until done
  B4    00EE    Return - End Fire Phasers Subroutine
```

DISPLAY NEW TARGET - ENTRY #1 WITH PHASER SIGHT INERTIA

```
03B6    8934    V9+V3 (target X+phaser sight movement from 0344-0354)
  B8    8A44    VA+V4 (target Y+   "      "       "      "   "   " )
  BA    23D4    Do subroutine - limit target XY to screen edges
  BC    CE01    VE=RND (random # between 0 and 1); ENTRY #2
  BE    4E00    Skips sight move while firing (50% of the time)

03C0    6EFF    VE=FF 50% of the time (03BC-03C0 chooses RND(1)
                or (FF)
  C2    89E4    V9=V9+VE : Add RND movement to target X
  C4    CE01    Repeat above 4 Lines for a new target Y
  C6    4E00
  C8    6EFF
  CA    8AE4
  CC    23D4    Do subroutine - limit target to screen edges
  CE    A5EA    I=target pattern

03D0    D9A4    Display @ V9, VA
  D2    00EE    Return - End Display New Target Subroutine
```

LIMIT TARGET TO SCREEN EDGES - Used only by Display New Target

```
03D4    49FF    Skip if V9 ≠ FF
  D6    6900    V9=0 (Left limit)
  D8    4939    Skip if V9 ≠ 39
```

```
03DA    6938    V9=38 (limit to right)
  DC    4A00    Skip if VA ≠ 0
  DE    6A01    VA=1 (limit to top is 01 so target firing
                doesn't wrap around)
03E0    4A3D    Skip if VA ≠ 3D
  E2    6A3C    VA=3C (limit to bottom of screen)
  E4    00EE    Return - End Limit Target XY subroutine
```

TARGET DESTRUCTION

```
03E6    6E01    VE=1 for tone
  E8    FE18    Sound tone (target hit)
  EA    A5EA    I=target pattern
  EC    6E0F    VE=0F - loop count
  EE    D9A4    Display @ V9, VA (03EE-03F6 is initial shudder
                before blow-up
03F0    D9A4    Display again to erase
  F2    7EFF    VE=VE+FF (loop count -1)
  F4    3E00    Skip if VE=0 (Done?)
  F6    13EE    Loop until done
  F8    7AFE    VA+FE (Target Y-2 for blow-up sequence)
  FA    A5C6    I=first destruct pattern
  FC    2418    Do subroutine to display debris
  FE    A5CE    I=second destruct pattern
0400    2418    Do sub - display debris
  02    A5D6    I=third destruct pattern
  04    2418    Do sub - display debris
  06    6900    V9=0 (0406-0414 selects one of 4 new target starts)
  08    6A01    VA=1
  0A    CE01    VE =RND # 00-01
  0C    3E00    Skip if VE=0 (50% of the time)
  0E    6938    V9=38 50% of the time
0410    CE01    VE=RND #00-01
  12    3E00    Skip if VE=00(50% of the time)
  14    6A3C    VA=3C 50% of the time
  16    00EE    Return - End Target Destruct subroutine
```

DISPLAY DEBRIS - Used only by Target Destruction subroutine

```
0418    D9A8    Display debris @ V9, VA
  1A    6E08    VE=8 for timer
  1C    2472    Do timer subroutine
  1E    D9A8    Display debris to erase
0420    00EE    Return - End Display Debris Subroutine
```

TARGET RETURNS FIRE

```
0422    6E04    VE=4 (loop count)
  24    A5EA    I=target pattern (0424-0426 enables the trans-
  26    D9A4    parent effect of the target's laser)
```

ADDRESS		CODE	COMMENTS
0428		7903	V9+03 (Adjust target X for laser position)
2A		7A01	VA+01 (Adjust target Y for laser position)
2C	*	A5DE	I=first pattern
2E		D9A2	Display @ V9, VA
0430		D9A2	Erase
32		79FF	V9+FF (X-1)
34		7AFF	VA+FF (Y-1)
36		A5E0	I=second pattern
38		D9A4	Display
3A		D9A4	Erase
3C		79FF	V9+FF (X-1)
3E		7AFF	VA+FF (Y-1)
0440		A5E4	I=third pattern
42		D9A6	Display
44		D9A6	Erase
46		79FF	V9+FF (-1) - reset target X
48		7A01	VA+1 - reset target Y
4A		7EFF	VE+FF (loop count -1)
4C		3E00	Skip if VE=0 (Done?)
4E		1424	Loop until done. Displays target between blasts of phaser fire for transparency effect.
0450		CE07	VE=RND #0-7
52		3E00	Skip if VE=0 (12.5% of the 3.125% times the target returns fire)
54		00EE	Return - End Target Returns Fire subroutine
56		0558	Do machine language subroutine - major hits on starship
58		7B01	VB=VB+1 (Number of hits +1)
5A		1454	Go to exit

DISPLAY THE SCORE

045C		A5C2	I=3-byte work area
5E		FD33	Convert VD (# of hits) to a 3-byte decimal
0460		F265	Load V0-V2 with the converted score @ I
62		651C	V5=1C (VX for score)
64		7609	V6+09 (VY for score, set by calling routine)
66		F129	I=bit pattern for V1 (V0 ignored; 30 is max)
68		D565	Display the first digit @ V5, V6
6A		7505	V5=V5+5 (VX)
6C		F229	I=bit pattern for V2
6E		D565	Display second digit @ V5, V6
0470		00EE	Return - End Display The Score Subroutine

* NOTE: Lines 042C-0444 seem to contain unnecessary redundancies. However, converting the repeated sections to a nested subroutine would only result in a 4-byte saving. This would be at the expense of speed (for CHIP-8 would spend extra time servicing calls) and would affect the realism of the target's laser - an effect that whould be greatly missed!

```
  TIMER SUBROUTINE
0472    FE15    Timer - value of VE (VE set by calling routines)
  74    FE07    VE=current timer value
  76    3E00    Skip if VE=0 (Done?)
  78    1474    Loop until timer=0
  7A    00EE    Return - End Timer Subroutine

  MESSAGE ROUTINES
047C    6702    V7=02(loop count) ; Title
  7E    6519    V5=19 (VX)
0480    6600    V6=0  (VY)
  82    A600    I=message
  84    0244    Call Messager
  86    D565    Display line
  88    760A    V6+0A (VY)
  8A    6514    V5=14 (VX)
  8C    0244    Call Messager
  8E    D565    Display
0490    760A    V6+0A (VY)
  92    650C    V5+0C (VX)
  94    0244    Call Messager
  96    D567    Display ("Wars" is 7 bytes deep)
  98    6639    V6=39
* 9A    6505    V5=5
  9E    D567    Display ( (c) line)
04A0    77FF    V7+FF (-1) Loop count
  A2    4700    Skip if V7≠0 (done?)
  A4    1302    Yes - go to exit
  A6    6EE0    VE=E0 for timer
  A8    2472    Do timer subroutine
  AA    147E    Loop to erase
  AC    6702    V7=2 (loop count) ; "You have 30 phasers"
  AE    A623    I=message
04B0    6618    V6=18 (VY)
  B2    650F    V5=0F (VX)
  B4    0244    Call Messager
  B6    D565    Display
  B8    760A    V6+0A (VY)
  BA    650C    V5=0C (VX)
  BC    0244    Call Messager
  BE    D565    Display
04C0    77FF    V7+FF (loop count -1)
  C2    4700    Skip if V7≠ 0 (Done?)
  C4    14CC    Yes - exit to next message
  C6    6E80    VE=80 for timer
  C8    2472    Do timer subroutine
  CA    14AE    Loop to erase
  CC    6702    V7=2 (Loop count) ; "Good luck!"
  CE    661E    V6=1E (VY)
```

```
* 0496  0244    Call Messager
```

```
04D0    A637    I=Message
  D2    0244    Call Messager
  D4    D565    Display
  D6    77FF    V7+FF (loop count -1)
  D8    4700    Skip if V7 ≠ 0 (done?)
  DA    1332    Yes - exit to check for next keypress
  DC    6E80    VE=80 for timer
  DE    2472    Do timer subroutine

04E0    14CE    Loop to erase
  E2    6702    V7=2 (loop count
  E4    A642    I=message  "Your Starship is Destroyed!"
  E6    6506    V5=6 (VX)
  E8    6616    V6=16 (VY)
  EA    0244    Call Messager
  EC    D565    Display
  EE    760E    V6+0E (VY for next line)

04F0    0244    Call Messager
  F2    D565    Display
  F4    77FF    V7+FF (loop count -1_
  F6    4700    Skip if V7 ≠ 0 (Done?)
  F8    1518    Yes - Go display score
  FA    6EA0    VE=A0 for timer
  FC    2472    Do timer subroutine
  FE    14E4    Loop to erase

0500    6702    V7=2 (Loop count)
  02    A697    I=message   "You Made it!"
  04    6508    V5=8 (VX)
  06    6616    V6=16 (VY)
  08    0244    Call Messager
  0A    D565    Display
  0C    77FF    V7+FF (Loop count -1)
  0E    4700    Skip if V7=0

0510    1518    Go display score
  12    6EA0    VE=A0 for timer
  14    2472    Do timer subroutine
  16    1502    Loop to erase
  18    6702    V7=02(Loop count)
  1A    A65E    I=Message  "You shot down (NN) Klingon Ships"
  1C    6506    V5=6  (VX)
  1E    6614    V6=14 (VY)

0520    0244    Call Messager
  22    D565    Display
  24    245C    Do Display Score subroutine
  26    6506    V5=6  (VX)
  28    7609    V6+9  (VY)
  2A    A66C    I=message
  2C    0244    Call Messager
  2E    D565    Display
```

```
0530    77FF    V7+FF  (loop count-1)
  32    4700    Skip if V7≠0 (Done?)
  34    153C    Go get next message.
  36    6EA0    VE=A0 for timer
  38    2472    Do timer subroutine
  3A    151A    Loop to erase
  3C    A67A    I=message    "Press Key "F" for another try"
  3E    6508    V5=8 (VX)
0540    6616    V6=16 (VY)
  42    0244    Call Messager
  44    D565    Display
  46    760E    V6+0E (VY for next line
  48    6502    V5=2 (VX)
  4A    0244    Call Messager
  4C    D565    Display
  4E    FF0A    VF=Key pressed

0550    3F0F    Skip if VF=0F
  52    154E    Loop until key F is pressed
  54    0230    Call Erase display routine
  56    1300    Go begin new game.  End Message Subroutine
```

MACHINE LANGUAGE SUBROUTINE

Starship Hit Sequence - 3K Systems. 4K System Owners see Modifications

```
0558    F8 0A AD 9B FC 02 BB F8
0560    08 BC 2C 9C 3A 62 9B FF
0568    02 BB F8 08 BC 2C 9C 3A
0570    6D 2D 8D 3A 5B D4
```

DATA STORAGE

```
05C2    0000    Work area for score
  C4    0000
  C6    0018    First target destruct pattern
  C8    4210
  CA    3670
  CC    8700
  CE    1481    Second target destruct pattern
05D0    2100
  D2    2003
  D4    6003
  D6    0020    Third target destruct pattern
  D8    0080
  DA    0001
  DC    00C1
```

```
05DE    C0C0    Target returns fire patterns
05E0    F0F0
  E2    F0F0
  E4    FCFC
  E6    FCFC
  E8    FCFC
  EA    1824    Target pattern
  EC    7EFF
  EE    8000    "Bip" for sight draw and phaser fire
05F0    1F00    V0 V1 (CHIP-8 variables storage area)
  F2    0000    V2 V3
  F4    0000    V4 V5
  F6    0000    V6 V7
  F8    0000    V8 V9
  FA    0000    VA VB
  FC    1E00    VC VD
  FE    0000    VE VF
```

MESSAGE STORAGE (In ASCII Code)

```
0600    6A6B    Title
  02    6C00
  04    5350    Space
  06    4143
  08    4500
  0A    7E60    -Wars
  0C    6162
  0E    6364
0610    6566
  12    677E
  14    0068    c  1979 T. Swan
  16    6931
  18    3937
  1A    3920
  1C    542E
  1E    5357
0620    414E
  22    0059    You have
  24    4F55
  26    2048
  28    4156
  2A    4500
  2C    3330    30 phasers
  2E    2050
0630    4841
  32    5345
  34    5253
  36    0047    Good Luck!
  38    4F4F
  3A    4420
```

```
063C    4C55
  3E    434B

0640    2100
  42    594F    Your Starship
  44    5552
  46    2053
  48    5441
  4A    5253
  4C    4849
  4E    5000

0650    4953    Is destroyed
  52    2020
  54    4445
  56    5354
  58    524F
  5A    5945
  5C    4400
  5E    594F    You shot down

0660    5520
  62    5348
  64    4F54
  66    2044
  68    4F57
  6A    4E00
  6C    4B4C    Klingon ships
  6E    494E

0670    474F
  72    4E20
  74    5348
  76    4950
  78    5300
  7A    5052    Press Key "F"
  7C    4553
  7E    5320

0680    4B45
  82    5922
  84    4622
  86    0046    For another try
  88    4F52
  8A    2041
  8C    4E4F
  8E    5448

0690    4552
  92    2054
  94    5259
  96    0059    You Made it!
  98    4F55
  9A    204D
  9C    4144
  9E    4520

06A0    4954
  A2    2100
```

MODIFICATIONS

4K systems use the following machine language routine changes for displaying a blinking screen when the starship is hit. See lines 0558-0574 in the Main Program Listing

055C F809

0566 9BF8
0568 0EBB

* PLEASE NOTE * PLEASE NOTE * PLEASE NOTE

4K systems load 11 pages from 0000 ("B" pages)
3K systems load 9 pages from 0000

64-BYTE CHECKSUM DATA FOR SPACE WARS (See VIPER, November, 1978, for instructions on using the following data.)

3K SYSTEMS		4K SYSTEMS	
byte check at:			
0300-BB	Row E3C3 AC51 Col 6877 1D12		Same as for 3K systems, except where noted.
0340-E6	Row AA20 D1C2 Col D4B8 653E		
0380-47	Row 33B1 22AF Col CC19 D41F		
03C0-40	Row 6DE3 B2F5 Col CD34 E6E4		
0400-61	Row 6B6E 8408 Col C838 1DA2		
0440-61	Row EA89 F9C7 Col BB2F A47F		
0480-60	Row D71C 440F Col 3D95 7F41		
04C0-D7	Row CDE2 1132 Col B5EB F4A7		
0500-F0	Row 035C 73EC Col 7022 E0B5		
0540-2E	Row 1F5A B930 Col 7057 8881	0540-36	Row 1F5D 2830 Col 6057 418A
0580	- - - - - -		
05C0-D3	Row A60B 7710 Col AA89 D6A3		

Load the checksum program (which is relocatable) at 0000. This eliminates the need to perform an initial (C0 0N 00) long branch, where N = the location of the checksum program.

Simply flip the run switch up, enter the above addresses, and check whether Space Wars is correctly loaded.

The last checksum (at 05C0) assumes that the scoring work area at 05C2 is set to zeros - as are all other unused memory locations.

Data for the ASCII coded messages @ 0600 is not included here, since this can be checked with any standard ASCII chart. All character strings are ended with a null character (00).

Data for the character set at 0700 was also not included. The characters may be examined using the Character Designer, by first relocating Space War's character set to 0600 (using the CHIP-8 Editor in chapter one), and then loading the Character Designer in 0000-05FF. If you decide to change any of the bit patterns, remember to re-relocate the new character set at 0700-08FF, then load 7 pages of Space Wars beginning at 0000. SAVE nine pages (which will include the character set changes) and then the program is ready to run.

If you are having troubles, and the checksums all check, make sure the modifications to the CHIP-8 Interpreter were done correctly. Use the 64 byte checksum data to check:

64 BYTE CHECKSUM DATA FOR THE MODIFIED INTERPRETER

(See VIPER, November, 1978, for instructions on using the data)

64-byte check at	*	0000-9A	Row 2A31 A484
			Col EC2A 1E9B
		0040-9A	Row CA77 EC91
			Col 62A4 7671
		0080-5B	Row 55A0 39E5
			Col 9A35 2B93
		00C0-A5	Row 7CAE 60E8
			Col A659 664A
		0100-5C	Row FB5D C951
			Col 05F5 2070
		0140-5D	Row 3862 02D6
			Col FEB0 7BBF
		0180-44	Row 092B 140D
			Col 5E1F BF45
		01C0-25	Row 4B6B BD00
			Col 5ACB D120
		0200-90	Row 55E4 94EA
			Col 0697 F215
	**	0240-19	Row C5F7 0629
			Col 8851 4C5E
	***	0280-25	Row 8F00 0000
			Col E061 8431

* First enter a C0 0A 00 long branch at 0000; load the checksum program at 0A00. When done, change 0000 back to 91 BB FF; and the interpreter is ready to run.

** With location 025E=07. See Message instructions.

*** All unused memory locations are set to zero.

S U R R O U N D

S U R R O U N D

Surround uses the normal (one page display) CHIP-8 Interpreter, with the program beginning at 0200.

Upon switching to RUN, the screen will be blank, and you must select one of two options. Key "1" will select a border to be drawn around the playing field, and key "2" will opt for full wrap-around with no border.

The object is to hit as many targets as possible. You control a snake-like trail (80 bits long) and can direct its forward movement by pressing Keys 2, 4, 6, or 8. The trail continues to move in the selected direction until you change it.

After reaching the maximum length, the "snake" begins to disappear from behind, but always remains at its full length from then on.

If you hit a target, you score 10 points. The score is displayed (yours on the left of the screen, the VIP's on the right). When the action restarts after a hit, the target is removed and your trail will start to move in the direction last selected.

If you hit anything other than a target (the border, for example, or your own trail), the computer scores 20 points. If you think this is unfair, see the instructions given later for modifying the program - but who said computer games have to be fair? At this point, the trail will "eat" itself backward to its beginning, the score is displayed, and the round starts over with a trail of zero length. Remember that the computer scores if you run into a wall or into your own trail...if you are going up and press the down key, you run into yourself! And the computer scores.

Whoever (you or the VIP) scores 100 points first is the winner. This is indicated by a flashing block over the winning score. Press Key "F" to restart; the screen will go blank and you

again have the option of selecting a border or a wrap-around game. I suggest you stick with the border until you get the hang of the controls.

Oh - and one small item I nearly forgot to mention. The longer the trail lasts, the faster it speeds up. Have fun!

S U R R O U N D

Although I prefer a subroutine structure, Surround was written in a straightforward manner. This was done for two reasons. First, the program isn't very long (only two pages), and it can be fairly easily comprehended. Second, there was a concern for timing problems. Several things happen between each "bip" of the trail, and subroutine calls (and returns) would only add a jerky movement to the snake. (This was the case with an earlier version, which was written with many subroutines.)

As a hint for your programming effort, sometimes it helps to write an entire program out of subroutine modules, which are then ordered by an executive program which calls each section in the proper sequence. The program can be left as it is, or, after debugging is complete, it can be tightened up. In fact, your CHIP-8 Interpreter is structured in this manner, with your program instructions providing the order of each function in the interpreter.

In Surround, memory locations 0200-0208 wait for either a Key 1 or Key 2 press to start the game. Key 1 activates the next section, which draws a border, while Key 2 sends the program directly to 022A. The border is drawn one bit short on two sides so the trail may run into it. Although this section could have been programmed more efficiently, I wanted the graphic effect of two lines drawing "at once" - admittedly a small point.

The next step is a machine language subroutine which copies the display page - with its border or without - into a separate memory page designated "0Z" and located two pages below the highest on-card RAM page. Like the Wizard of Oz, display page "0Z" is a faker, and is used solely to display the score. This was done in order to give the appearance of the score being displayed on the same display page as the game. You

don't notice the switch (performed by two other machine language subroutines at 03BE and 03C4) because the border is the same for both. The only reason for the routine is to allow fast restart of the trail after hitting a target. The trail appears to disappear, the score comes on, the entire trail returns and the game restarts. Actually, the trail is always there. You're simply looking at a display that always has a border (or for a wrap-around game, a non-border). It's a nice effect that could be used for many other games! (I haven't detailed the machine language routines, but they're simple enough, and should be easy to figure out.)

After initializing all variables with lines 022C and 022E, the game begins. I always include this initialization procedure, keeping the starting values for each variable in a data array - at 03F0, in this case. It then becomes a simple matter not only to change initial values, but more importantly, to add a new variable which wasn't thought of when I started.

Locations 0230-023C cycle V0 one time through values 2, 4, 6 and 8 to test if the key is pressed. If one has been pressed, its value is in V0; then V3 and V4 are set to zero. These two variables will later be added to the coordinates of the trail, to create the new position, and to give the appearance of movement. If no keys are pressed, then V3 and V4 do not change and the trail will continue in the direction last selected.

Locations 0242-0254 select new values for V3 and V4. The values are then added to the trail's X and Y coordinates: V8 and V9. This section is cycled through, even if no key is pressed, for the timing reasons mentioned earlier; however, the jump at 023A could be to 0252 with only a small change in effect.

Control now passes to a routine at 02A6. We'll discuss this later (it's rather complex and we'd get side-tracked from here), although its function is to record the XY coordinates of each

trail bip to allow for erasing its "behind" and for resetting if (when!) you run into the wall.

Continuing at 0258, the next trail bip is displayed, a tone is sounded, and a test is made to determine if the trail hit anything (if VF=1). If so, control passes to the scoring routines at 02CA, which determine if a target was hit, check for a win, display the score, etc....More on this routine a little later (when we get to it in the natural sequence of things).

If the trail didn't hit anything, the next section determines the speed of the trail. (Line 0264 is a "no operation" instruction which replaces a "random speed increase" that didn't work out very well.) Every time the trail cycles through one 80-bip distance (indicated by V7=9E), the speed is increased by subtracting 2 from V5. The maximum speed that the trail can attain is set in line 026A, which may be changed to 35KK, in which KK is any even number in the range 00-0C. Try 3504 or 3506 if you think the trail moves too fast; if you try 3500, the trail will eventually shift into full warp drive.

Lines 026E and 0270 demonstrate the use of a helpful timing techniques I use in all my game programming. Turn to line 038C. You'll see a simple timing subroutine that sets the CHIP-8 timer equal to the value of VE, tests to see if the timer equals zero yet, and only returns via an 00EE instruction afterwards. (The timer, remember, automatically begins to decrement the instant it is set, so the same variable (VE in this case) can be used to see when it goes to zero, so long as the value doesn't have to be preserved.) I use this subroutine in many ways - which is a good reason to have it in subroutine form - and only have to set a value into VE, call the routine with a 2MMM instruction, and the program will halt where it is for a value relative to VE!

Getting back to 026E: You'll see that VE is set equal to V5 (the trail-speed indicator) and thetiming subroutine is called.

As V5 decreases, the amount of time spent in the time loop is also decreased, and the trail speed increases - right on through the ceiling (or, as is more likely, one of the four walls, and 20 points for you-know-who!)

The next section, beginning at 0272, controls the target placement. First a random number is set into VE and the number is tested to see if it's equal to zero (for a decision whether to display a new target on the screen). The percentage of time a target stays where it is may be changed with a new mask at line 0272. Remember, though, you may only use numbers with the CXKK instruction that contain unbroken bit positions (speaking binarily) from right to left. If you are confused by all this, write down - in binary - the following numbers: 01, 03, 05, 07, 0F, 3F, 5F, 7F, FF, and you'll see what I mean. The number 05, for example, won't work because it contains binary "holes" (o's) which defeat it's ability to mask the random number. Changing 0272, then, to CE1F will permit the target to remain on the screen a longer time. As the target is sometimes off, however, this will also result in its remaining off for a longer time. Thus, by trying to make the game easier, you may make it harder. (I told you computers weren't fair.)

If the decision is to display a target, the "target on" flag (VA) is tested to see if there is an old target already being displayed. (Replace the instruction at 027A with a 127C; GO TO 027C instruction; and you'll see why the flag is necessary.)

Lines 027E - 0298 choose new random XY coordinates for the target and limit these to the screen edges. This keeps the target off the borders if there are any, and eliminates target wrap-around, if there aren't any borders. It also keeps the target in a testable range for checking to see if it was hit, although this could be done in other ways. (Masking the X coordinate by a logical "AND" with 3F, for instance.)

The program then displays the target, checking to see if it ran into anything. Obviously, the trail can be anywhere

at any time, and the target can only go in empty space. So if it hits something (i.e., the trail), it is immediately erased at line 02A0, the target flag is reset, and the game continues without a target on display until the VIP decides to create a new one. This will occasionally result in a flash and a momentary jerk of the trail, but this is hardly objectionable. A routine test to see if the location is vacant could be written, which would be invisible to the game player. Execution of such a routine would take time, however, and would be best done in machine language. The results don't seem to me to be worth the trouble.

And so ends the main program loop, with return instructions at 02A4 and 0276. The following section describes the XY coordinate control, target hit, and scoring routines (as I promised we would get to in due time).

Lines 02A6-02C8 contain one of the more interesting routines of the program. V7 is a memory index which is added to the value of I (initialized to 0400), and then incremented by 2 for each new pass. This will form an upward growing stack (or array) for storing successive XY trail coordinates in memory with the F155 instruction at 02B0. The trail length will occupy two bytes for each of its "bips". A trail of 80 bips will need memory space from 0400-04BF, and the largest possible trail (128 bips) would fill the memory page. V7, then, determines the length of the trail, and when V7 is at its maximum value, the section at 02BE is enabled (by setting flag V6=1). This section erases the bips from behind. The memory index V7 is reset to 0, and will therefore limit the size of the storage array to only the length needed. Since the index starts over when the trail reaches maximum length, the array appears circular, with the last position erased just two bytes ahead of the first bip on the trail.

The size of the trail may be altered by changing 02B2 to skip when V7 reaches the desired trail length times two. (This,

of course, must be an even number.) However, the trail speed uses V7 to decide when to speed up, as previously described. This will have to be adjusted if you program a shorter trail, or the trail will never speed up. With imagination, you might want to program a variable-length trail, but I leave this as an exercise to the reader.

When the computer scores, the trail erases in reverse. If you change the size of the trail, you'll also need to change locations 0316, 031C, and 033A to reflect the changes. These locations are well marked in the program listing. 0316 and 031C use a value of the trail length minus two, while 033A uses the value of the exact length. To be exactly correct, the trail is always one bip shorter (two bytes of storage) than I have said it is, since a bip is being taken off each time a new bip is being created. So while the trail is 80 bips long in memory, it is 79 bips long on display.

Locations 02CA-02F6 determine if a target was hit. The trail XY coordinates must first be masked to the effective range of the target with logical "AND" instructions. (This is only in the wrap-around version, where the trail may hit a target while having different XY coordinates.) VX=0 and VX=40 are at the same screen location. 40 AND 3F will equal zero. Remember, the target XY coordinates were limited to this range earlier in the program. The routine here simply loops through all the possible XY coordinates (9) of the target after a hit is detected. If any of these match those of the trail, then a target was hit and the user scores. If not, the computer scores. The result of the test is indicated by V0=1, a temporary flag, rather than simply exiting from the test. This allows the target's XY coordinates to be easily reset in order to erase the target. (On second thought, the target could have been erased first, then the check made. This proves my theorem than any program longer than zero bytes can be shortened. The most efficient size for any program must therefore be equal to zero. But it's only a theory.)

We come now to the scoring routines, which don't need to be covered here. The VIP manual covers the process of converting and displaying numbers on the screen, and I haven't done any new things to the routines. The only unusual part is the change of display pages as previously discussed. But notice the use of my timer subroutine again.

Locations 0310-0342 erase the trail from front to back when you run into something other than the target. This is a more complex version of the memory array indexing system used to store the coordinates, but it works basically the same way. In reverse. Three possible trail conditions must be tested:

1) Trail not at maximum length
2) Trail over maximum length
3) Trail just reached maximum length

At each, the memory pointer must be treated specifically to point to the right information and erase no more than the number of bips on the screen. Since the array is circular, the index must be carefully controlled.

Modifications And Helpful Hints

Locations 0352-0358 test for a win, and can be altered to allow higher or lower end-game scores.

The initial trail speed is set by the value in the data array at 03F5. This can be switched to any even number for a new starting trail speed. Since the trail speed is increased by a factor of two, even numbers are necessary here.

The VIP manual makes the statement that 02 is the smallest values that the speaker will respond to, and that sounding the tone for a unit of 01 will not work. I have found this to be untrue, and prefer the shorter "beep". You should hear a beep every time the trail moves, and a fast sequence of seven beeps when a target is hit. If you don't, try changing 02FD and 025D from 01 to 02 for a slightly longer beep.

Since VA (new target set flag) is always FF after a scoring routine (because the target is off), I use VA to test for a press of Key "F" to restart the game. (The restart feature is at location 0374.) "Tricks" like this can cause bugs, and it's better to initialize a variable before it is used rather than assuming it has any particular value in it. (The CHIP-8 Interpreter does this fairly often, and could be criticized for it - although, for the extreme condensation of the language, this and other "dangerous tricks" were necessary and desireable tradeoffs.)

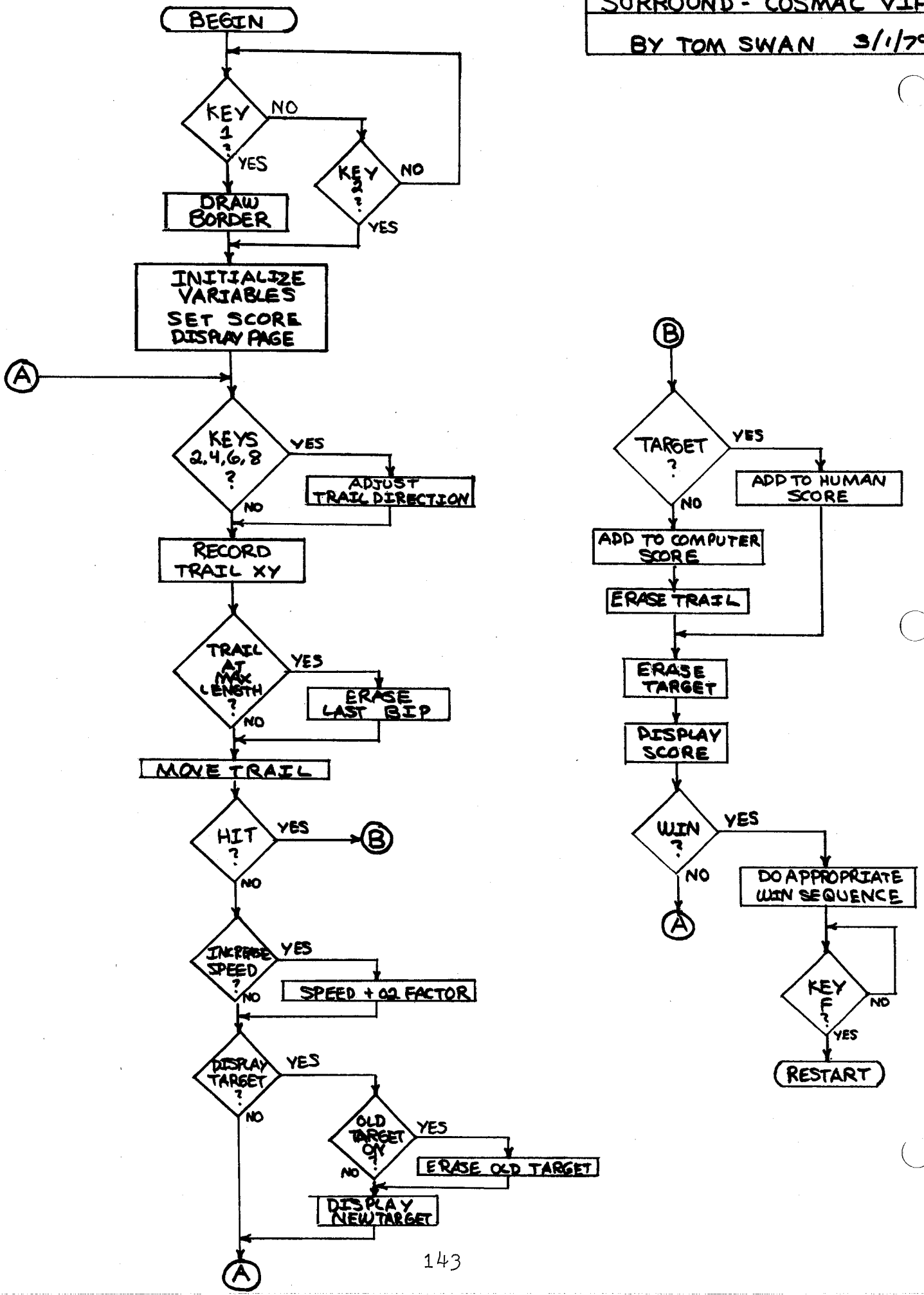
Because the game is over at this point, I think I'm safe, but a later change in the use of VA could disable the restart feature. A general rule for your programming is to keep the individual parts as loosely connected as possible - don't share values of things or set parameters in one routine for use in another. Your program may not run any differently, but the debugging process will go more smoothly. For instance, I usually set "I" with an AMMM instruction immediately prior to a DXYN (display) instruction, even though it may have been set properly by a previously routine. This is because I might someday (when I'm revamping the program, for example) want to jump into this part of the program from another routine which does not leave "I" properly set...and I don't want to have to chase down the "garbage" that might appear on the screen while I'm busily being creative!

Just in case you have any trouble, I have included checksum data to help you determine whether the code is properly entered. See the VIPER, November, 1978, for instructions on how to use the checksum data. The program must be entered exactly as listed, with all unused memory locations set to zero except where noted with NOP (no operation) lines.

I hope you enjoy Surround - and are willing to get your fingers a little "dirty" by experimenting with some of the mods I've suggested. Have fun!

SURROUND - COSMAC VIP

BY TOM SWAN 3/1/79



S U R R O U N D

VARIABLE ASSIGNMENT

V0 - Key Presses - Utility etc..
V1 - } Used to Display Score
V2 - Utility Erase Trail Routine
V3 - Trail X Coordinate Direction - Initially = 02
V4 - Trail Y Coordinate Direction - Initially = 00
V5 - Trail Speed - Initially = 0C
V6 - Trail Erase Flag - Initially = 00
V7 - Trail Length/Erase Index - Initially = 00
V8 - X Coordinate Trail - Initially = 02
V9 - Y Coordinate Trail - Initially = 1C
VA - X Coordinate Target - Initially = FF - Also first target flag
VB - Y Coordinate Target - Initially = Don't care
VC - Human Score - Initially = 00
VD - Computer Score - Initially = 00
VE - } Utility Loops etc..
VF - }

ROUTINE LOCATIONS

0200 - 03FF -Program
0400 - 04NN -Trail storage where NN = length of trail x 2
0X00 - 0XFF -Display page - X = highest page on card RAM
0Z00 - 0ZFF -Display score page - 0Z = Display page - 02
0500 for 2K, 0900 for 3K, 0D00 for 4K

SUBROUTINES

038C - Timer
0396 - Display score
03BE - MLS - Score page change
03C4 - MLS - Regular page change
03CA - MLS - Copy to score page

S U R R O U N D

PROGRAM LISTING

```
0200 - FF0A - VF=Key press - wait
  02 - 4F02 - Skip VF≠02
* 04 - 122A - Go no border
  06 - 3F01 - Skip VF=01
  08 - 1200 - Go loop for correct key
  0A - A3E7 - I=bit for border
  0C - 6000 - V0=00  X Coordinate
  0E - 6100 - V1=00  Y Coordinate -top
0210 - D011 - Display @ V0 V1
  12 - 611E - V1=1E -bottom
  14 - D011 - Display @ V0 V1
  16 - 7001 - V0+01
  18 - 303F - Skip V0=3F -end lines
  1A - 120E - Go loop top/bottom lines
  1C - 603E - V0=3E right
  1E - 71FF - V1+FF(-01)
0220 - D011 - Display @ V0 V1
  22 - 6000 - V0=00 left
  24 - D011 - Display @ V0 V1
  26 - 3101 - Skip V1=01
  28 - 121C - Go loop right/left lines
  2A - 03CA - Do MLS @ 02CC - Copy display into score page
  2C - A3F0 - I=storage array variables
  2E - FF65 - Load V0-VF with data @ I
0230 - 6000 - V0=00 for key press check - begin program
  32 - 7002 - V0+02
  34 - E0A1 - Skip V0≠key pressed
  36 - 123E - Go move trail
  38 - 400A - Skip V0≠0A (Done one cycle)
  3A - 1242 - Go move trail - no key selected
  3C - 1232 - Go loop till key 2;4;6;8 pressed/or done
  3E - 6300 - V3=00 - Move direction adders
```

*NOTE- border is drawn one bit short on the right and bottom to permit the trail to hit it.

```
0240 - 6400 - V4=00 - Move direction adders
0242 - 4002 - Skip V0≠02
  44 - 64FE - V4=FE -Up
  46 - 4004 - Skip V0≠04
  48 - 63FE - V3=FE -left
  4A - 4006 - Skip V0≠06
  4C - 6302 - V3=02 -right
  4E - 4008 - Skip V0≠08
0250 - 6402 - V4=02 -down
  52 - 8834 - V8+V3 -adjust trail X
  54 - 8944 - V9+V4 -adjust trail Y
  56 - 12A6 - Go record trail XY
  58 - A3E7 - I= trail piece
  5A - D891 - Display @ V8 V9
  5C - 6E01 - VE=01
  5E - FE18 - Sound tone for VE (trail bip)
0260 - 4F01 - Skip VF≠01 -No hit
  62 - 12CA - Go 02CA - trail hit something
  64 - 1266 - Go -(No operation)
  66 - 379E - Skip V7=9E  (Every 80 bips of trail)
  68 - 126E - Go no speed change
  6A - 3502 - Skip V5=02 -Maximum speed
  6C - 75FE - V5+FE (-02) Increase speed
  6E - 8E50 - VE=V5 For time loop
0270 - 238C - Do sub -time loop for trail speed
  72 - CE0F - VE=RND # 00-0F-Decide to display target
  74 - 3E00 - Skip VE=00
  76 - 1230 - Go loop -no new target
  78 - A3E4 - I= target pattern
  7A - 3AFF - Skip VA=FF/ First target enable flag
  7C - DAB3 - Display to erase old target
  7E - CA3F - VA=RND # 00-3F  X Coordinate target
0280 - CB1F - VB=RND # 00-1F  Y Coordinate target
  82 - 7A01 - VA+01 (VA must not= 00 -left border limit)
  84 - 7B01 - VB+01 (VB  "    " " "  -top   "       "  )
  86 - 80A0 - V0=VA  to test
  88 - 81B0 - V1=VB   "   "
  8A - 6E3C - VE=3C (Right border limit)
  8C - 80E5 - V0-VE
  8E - 4F01 - Skip VF≠01 (Not Greater than or Equal to 3C)
0290 - 6A3B - VA=3B (Limit VA to LESS than 3C)
  92 - 6E1C - VE=1C (Bottom border limit)
  94 - 81E5 - V1-VE
  96 - 4F01 - Skip VF≠01 (Not Greater than or Equal to 1C)
  98 - 6B1B - VB=1B (Limit VB to LESS than 1C)
  9A - DAB3 - Display target
  9C - 3F01 - Skip VF=01  Target hit something
  9E - 1230 - Go loop main
02A0 - DAB3 - Display to erase invalid target
```

```
02A2 - 6AFF - VA=FF  Set new target enable flag
  A4 - 1230 - Go loop main
  A6 - A400 - I= memory arrary - Trail XY coordinate storage/trail control
  A8 - F71E - I= I+V7 (Index)
  AA - 7702 - V7+02 For next time through and for erasing last bips
  AC - 8080 - V0=V8 For next store instruction
  AE - 8190 - V1=V9  "   "     "       "
02B0 - F155 - Store V0 V1 @ I (2 bytes each trail location)
  B2 - 37A0 - Skip V7=A0- memory full - trail at max. first time through
  B4 - 12BA - Go skip erase & reset index
  B6 - 6601 - V6=01 - erase trail bips flag
  B8 - 6700 - V7=00 - reset trail length index
  BA - 3601 - Skip V6=01 erase flag set
  BC - 1258 - Go 0258 -end trail routine part one
  BE - A400 - I= memory array trail coordinates
02C0 - F71E - I= I+V7  Index to last trail bip position
  C2 - F165 - Load V0:V1 with coordinates of last bip in trail
  C4 - A3E7 - I= bip pattern
  C6 - D011 - Display to erase
  C8 - 1258 - Go 0258 -end trail routine part two
  CA - 6F3F - VF=3F  Enter here- trail hit something
  CC - 88F2 - V8 & VF (Mask to 3F)
  CE - 6F1F - VF=1F
02D0 - 89F2 - V9 & VF (Mask to 1F) Limits trail XY to target range
  D2 - 6000 - V0=00 -Set a flag =00
  D4 - 6F03 - VF=03 Loop count #2
  D6 - 6E03 - VE=03 Loop count #1
  D8 - 5A80 - Skip VA=V8 (X coordinates trail/target equal)
  DA - 12E0 - Go -check next X
  DC - 9B90 - Skip VB≠V9 (Y coordinates NOT equal)
  DE - 6001 - V0=01 Set flag - target was hit
02E0 - 7A01 - VA+01 Next bit in row
  E2 - 7EFF - VE+FF (-01)  Loop #1 -01
  E4 - 3E00 - Skip VE=00  Done loop
  E6 - 12D8 - Go check to end of row
  E8 - 7AFD - VA+FD (-03)  Reset target X
  EA - 7B01 - VB+01  For next row in target
  EC - 7FFF - VF+FF (-01)  Loop #2 -01
  EE - 3F00 - Skip VF=00  Done loop
02F0 - 12D6 - Go check all rows
  F2 - 7BFD - VB+FD (-03)  Reset target Y
  F4 - 3001 - Skip V0=01  Target hit flag set
  F6 - 1310 - Go computer scores
  F8 - 7C0A - VC+0A (Human score +10)   Human scores enter here
  FA - 6F07 - VF= Loop count for beeper
  FC - 6E01 - VE= 01 for tone
  FE - FE18 - Sound tone for VE
0300 - 7FFF - VF+FF (-01) Loop count
  02 - 6E05 - VE=05 For time loop sub
```

```
0304 - 238C - Do sub  Time loop
  06 - 3F00 - Skip VF=00  Done loop
  08 - 12FC - Go loop till done
  0A - 130E - Go (No operation)
  0C - 0000 - Go (No operation)
  0E - 1346 - Go Scoring routines
0310 - 7D14 - VD+14 Computer score +20/Computer scores enter here
  12 - 6E20 - VE=20
  14 - FE18 - Sound tone for VE
  16 - 629E - V2=9E (Max. trail length -02)
  18 - 3700 - Skip V7=00 (Index is = base address)
  1A - 1322 -
  1C - 679E - V7=9E -Sets V7= Max. trail -02 when V7=00
  1E - 6600 - V6=00 -Disable erase flag
0320 - 1324 - Go skip next
  22 - 77FE - V2+FE (-02)  Index in reverse
  24 - A400 - I= base address trail coordinates storage
  26 - F71E - I+V7
  28 - F165 - V0 V1= Data @ I
  2A - A3E7 - I= trail bip pattern
  2C - D011 - Display to erase
  2E - 6E02 - VE=02 for time sub
0330 - 238C - Do Time sub- (Graphics control)
  32 - 3700 - Skip V7=00
  34 - 133C
  36 - 3601 - Skip V6=01 (Erase enable set)
  38 - 1342 - Go Scoring (Done)
  3A - 67A0 - V7=A0 -Maximum length
  3C - 72FE - V2-02 (Loop count when needed -02)
  3E - 3200 - Skip V2=00 (Done)
0340 - 1322 - Go loop
  42 - A3F0 - I= storage array for variables
  44 - F965 - V0-V9= Data @ I  Reset needed variables
  46 - A3E4 - I= Target pattern (Enter here from human scores)
  48 - 3AFF - Skip VA=FF -Target enable (Skips when no target on screen)
  4A - DAB3 - Display to erase target
  4C - 03BE - Do MLS- Change to score display page
  4E - 2396 - Do Sub - Display score
0350 - 6AFF - VA=FF  Reset new target enable flag
  52 - 4C64 - Skip VC≠64 (Check for human win)
  54 - 1364 - Go Human wins
  56 - 4D64 - Skip VD≠64 (Check for computer win)
  58 - 1368 - Go Computer wins
  5A - 6E90 - VE=90 Timer value for score display
  5C - 238C - Do Sub- Timer
  5E - 2396 - Do Sub- Display score to erase
0360 - 03C4 - Do MLS- Reset to normal display page
  62 - 1230 - Go Continue
  64 - 6009 - V0=09  Human wins block
```

```
0366 - 136A - Go Display win block
  68 - 601F - V0=1F Computer wins block
  6A - 6105 - V1=05 Win block Y coordinate
  6C - A3E8 - I= Pattern for block
  6E - 8200 - V2=V0 (Preserves V0 for blink)
0370 - 6303 - V3=03 Loop count
  72 - D217 - Display @ V2 V1
  74 - EAA1 - Skip VA ≠ keypress (VA always = FF here)
  76 - 1386 - Go restart new game
  78 - 7207 - V2+07
  7A - 73FF - V3+FF (-01)
  7C - 3300 - Skip V3=00 Loop Done
  7E - 1372 - Go loop - Display full block
0380 - 6E10 - VE=10  Timer value
  82 - 238C - Do Timer Sub (Waits between blocks)
  84 - 136E - Go loop for a new block
  86 - 03C4 - Do MLS Reset Display page
  88 - 00E0 - Erase display
  8A - 1200 - Go 0200 for new game
  8C - FE15 - Timer= VE (Set by calling routines) - Timer Sub
  8E - FE07 - VE=Current timer value
0390 - 3E00 - Skip VE-00 (Done)
  92 - 138E - Loop till done
  94 - 00EE - Return
  96 - 6B0C - VB=0C Score X coordinate  }  Begin display score sub
  98 - 6E06 - VE=06 Score Y coordinate  }  only two available
  9A - A3E0 - I= work area (3 bytes) for score
  9C - FC33 - VC=3DD# @ I
  9E - 23AA - Do Sub- display the score (Human's) -3 numbers sub
03A0 - 7B0C - VB+0C  For computer score- spaces the two scores
  A2 - A3E0 - I= Work area (same)
  A4 - FD33 - VD=3DD# @ I
  A6 - 23AA - Do sub- Display the score (Computer's) -3 numbers sub
  A8 - 00EE - Return
  AA - F265 - V0-V2= Data (Score @I) -Begin display 3 numbers sub
  AC - F029 - I= Pattern in ROM For V0
  AE - DBE5 - Display @ VB VE  First digit
03B0 - 7B05 - VB+05
  B2 - F129 - I= Pattern for V1
  B4 - DBE5 - Display @ VB VE  Second digit
  B6 - 7B05 - VB+05
  B8 - F229 - I= Pattern for V2
  BA - DBE5 - Display @ VB VE  Third Digit
  BC - 00EE - Return
  BE - 019B - MLS #1 (01 is NOP marker)  Set to score display page
03C0 - FF02
  C2 - BBD4
  C4 - 029B - MLS #2 (02 is NOP marker) Set to regular display page
  C6 - FC02
```

```
03C8 - BBD4
  CA - 039B - MLS #3 (03 is NOP Marker)  Copy contents regular display page
  CC - BCFF                              into score display page (duplicate
  CE - 02BD                              either a border or a no border fie
03D0 - F8FF
  D2 - ACAD
  D4 - ED0C
  D6 - 732C
  D8 - 8CFB
  DA - FF3A
  DC - D5D4
  DE - 0000
03E0 - 0000 - Scoring work area
  E2 - 0000 -   "      "    "
  E4 - E0E0 - Target
  E6 - E080 - and trail pattern
  E8 - FEFE - Win block pattern
  EA - FEFE -  "     "      "
  EC - FEFE -  "     "      "
  EE - FE00 -  "     "      "
03F0 - 0000 - V0 V1  Chip-8 Variables data storage
  F2 - 0002 - V2 V3    "        "        "       "
  F4 - 000C - V4 V5    "        "        "       "
  F6 - 0000 - V6 V7    "        "        "       "
  F8 - 021C - V8 V9    "        "        "       "
  FA - FF00 - VA VB    "        "        "       "
  FC - 0000 - VC VD    "        "        "       "
  FE - 0000 - VE VF    "        "        "       "
```

64 BYTE CHECKSUM DATA FOR SURROUND

See the November, 1978 VIPER for instructions on using this data.

64-byte check at	0200-B0	Row 1790 E18D
		Col 6427 75AA
	0240-A2	Row 4EC6 78C0
		Col 4F3C 767B
	0280-82	Row 1328 72A1
		Col 23C3 FF9F
	02C0-11	Row 1452 47B2
		Col AFC9 39BA
	0300-73	Row A9B2 178D
		Col D215 2E45
	0340-06	Row 289E C118
		Col 23CF CA01
	0380-1A	Row 3AC2 7051
		Col 2045 0E97
	03C0-60	Row FC55 41EF
		Col 4720 8077

Load the checksum program (which is relocatable) at 0000. This eliminates the need to perform an initial (C0 ON 00) long branch, where N=the location of the checksum program. Simply flip the run switch up, enter the above addresses, and check to see that Surround is loaded correctly.

The last checksum at 03C0 assumes that the scoring work area at 03E0 is initially set to zeros - as are all unused memory locations.

A P P E N D I X A

Modifications to the ELF II - to run
CHIP-8 and other VIP games

by Bobby R Lewis

MODIFY THE ELF-II TO RUN CHIP-8 AND VIP GAMES

by Bobby R. Lewis

(Editor's note: We have received requests from literally dozens of VIPER readers requesting information on how they can use their ELF-IIs to run VIP games in CHIP-8. Since the VIPER is dedicated to users of the VIP rather than to users of the ELF-II, we haven't exactly been what one would call responsive to the inquiries. One reader wrote: "Most of the more elaborate games, etc. seem to have been written in CHIP-8, and at my present level of skill, I'm not up to decoding and re-writing RCA's VIP monitor and CHIP-8 to run on my Super Elf. The manual that came with my machine is confined to very elementary stuff - besides that, it's terrible. They don't know a thing about communication. Help us, VIPER people! We need a "liberator"! " This particular reader reflects the same aura of "desperation" we've seen in several dozen similar letters, and we've been scratching our heads about how such an article would fit into the "exclusively VIP" format we've adopted.

A careful look at Bobby Lewis's article convinced us that the information will be useful to many of our subscribers, even if the idea is to use an ELF II rather than a VIP to start with. However, once the modifications have been made, the user will have an "effective" VIP - and will be among the exclusive membership of the VIPER subscribers! That, to us, was enough of an excuse to justify inclusion of the article in this issue. So here it is - enjoy!)

If you own an ELF-II, you can take advantage of CHIP-8 and RCA VIP games. Although the following information assumes that you have an ELF-II with Giant Board Monitor and at least 4K of RAM, you can possibly use it on other systems if you are familiar with the VIP operations and hardware.

First, let's take a look at some of the VIP features. A basic system contains a 512 byte (2 page) operating system in ROM, addressed at 8000 through 81FF. The operating system normally searches for and uses the highest page of RAM for the operating system display page. The VIP manual, contains a hexadecimal listing for the 512 byte CHIP-8 interpreter that must be entered into addresses 0000 through 01FF with the hex keypad. The VIP has no 7-segment displays for data; instead, it uses the video screen in conjunction with the operating system to display addresses and data and to allow modification of the data. The keyboard on the VIP is what I like to call

a "dynamic" keyboard. It is perfectly suited to interactive programs and games that require keyboard responses. This is a very simple design to implement on the ELF=II, and full details are included in this article.

Before you start thinking about using the original ELF-II key board, you must be aware of the fact that it latches up 8 bits on the data bus when INPUT is pressed. In addition, the low (or first) digit pressed is shifted when you enter a subsequent digit. Even if you write another routine to read the ELF-II keyboard, you will be pressing three keys to do the function of one on a dynamic-type keyboard. This situation won't allow you to take full advantage of the CHIP-8 games.

Before I discuss the actual conversion, refer to figure 1 for a summary of ELF-II and VIP I/O instructions. The 64 instruction is really doing the same function on both machines, but with the ELF-II, you have the additional feature of displaying the contents of memory on the 7-segment displays. The other mahor difference between the I/O instructions is the hex keypad enables. The ELF-II uses the 6C, which is an input instruction, and the VIP uses a 62, which is an output instruction. This should give you a clue as to why a different keyboard is needed to run CHIP-8.

ELF-II	VIP
61 video off	61 video off
62 available	62 output to keyboard
63 available	63 output port
64 display LEDs	64 Mx-bus, Rx 1
65 available	65 available
66 available	66 available
67 output port	67 available
68 illegal	68 illegal
69 video on	69 video on
6A available	6A available
6B available	6B input port
6C input from keyboard	6C available
6D available	6D available
6E available	6E available
6F input port	6F available

FIGURE 1
I/O INSTRUCTION SUMMARY

The operating system in the VIP actually outputs the low 4 bits of the data bus to a (4 to 16 line decoder) attached to one side of the hex keypad, allowing EF3 to be enabled corresponding to the key being pressed. So actually, the VIP keypad only inputs EF3, not data. The ELF-II, on the other

hand, latches up 8 bits, 4 bits at a time, and inputs this information to the data bus when the input switch (EF4) is pressed. Figure 2 contains a summary of the EF flag usage for both systems.

ELF-II	
EF1 video interface	EF1 video interface
EF2 cassette interface	EF2 cassette interface
EF3 available	EF3 hex keypad
EF4 hex keypad	EF4 available

FIGURE 2 - EF FLAG USAGE

HARDWARE MODIFICATIONS

Refer to figure 3 for details to implement a VIP type dynamic keypad to your ELF-II. The keyboard circuit can be hooked directly to the data bus, bit it is easier to use the existing output port. The numbers in () are actual pin numbers at the output port on the Giant Board. Although a CD4514 should be used, I used the TTL equivalent (74154) with no problems. Since pins 6, 7, and 8 are not presently used on the output port, you can jumper +5V, ground, and EF3 or EF4 to them from anywhere on the Giant Board. Use EF4, if not already in use, and you already have a diode for use at the hex/term switch location. These are the only hardware modifications required to your ELF-II

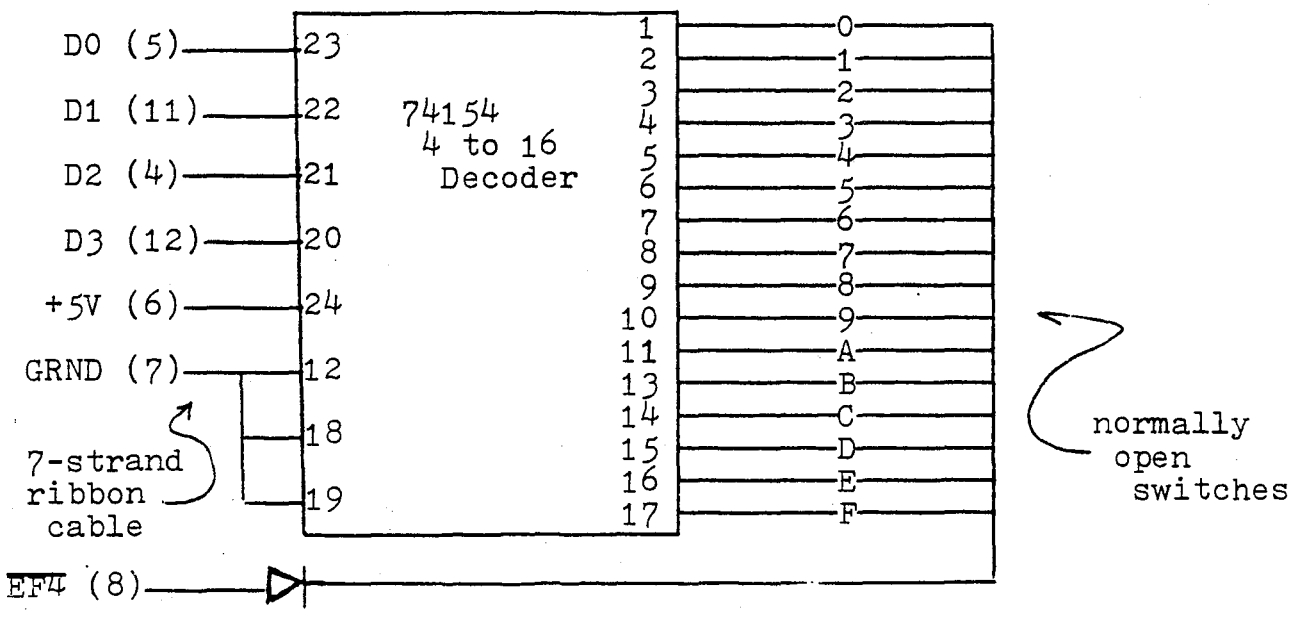
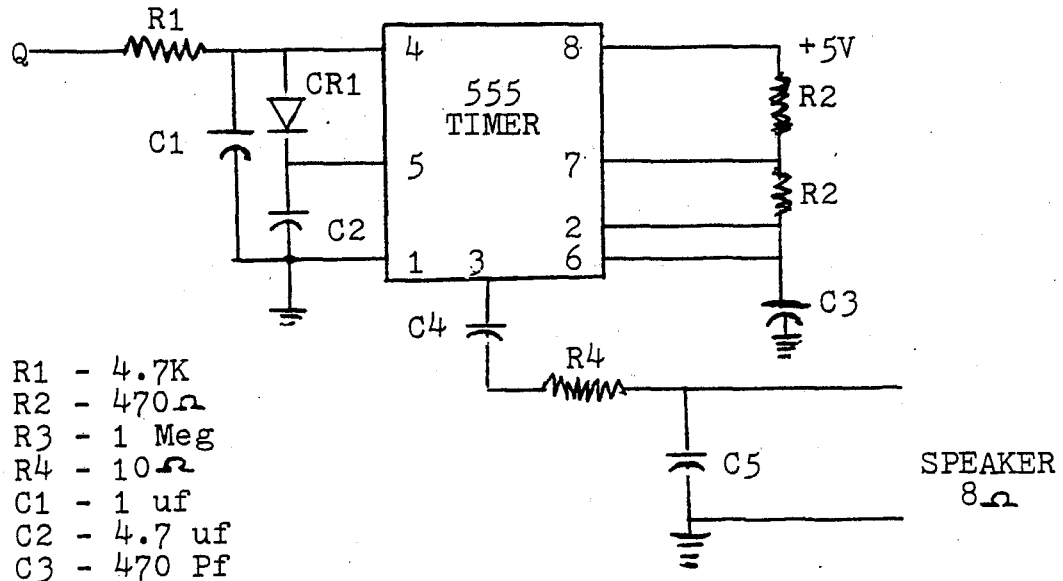


FIGURE 3 - HEX KEYBOARD

For a keyboard, I used a small CASIO calculator keyboard, because I like the feel of the keys. Any 16 key matrix will do. I used a 7-strand cable about three feet long, with a 14-pin dip header to plug into my Giant Board.

As a precaution, it is recommended that a +5V regulator be added to the Giant Board in the space provided, to ensure that you don't overload the motherboard regulator.

Many of the VIP games have a "beep" tone built into them, so I've included the tone circuit to be added to Q if you want the tone. See figure 4 for the circuit, and note the modifications to the VIP circuit.



R1 - 4.7K
R2 - 470Ω
R3 - 1 Meg
R4 - 10Ω
C1 - 1 uf
C2 - 4.7 uf
C3 - 470 Pf
C4 - 22 uf
C5 - 1 uf
CR1- IN914

Note: Modification of VIP circuit

FIGURE 4 - TONE CIRCUIT

SOFTWARE MODIFICATIONS

First, we'll discuss the changes to be made to the operating system. The OS for the VIP is contained in ROM at addresses 8000 through 81FF. Although this code could be loaded at any page boundary, we'll put it in address 0D00 through 0EFF. You will need to obtain a copy of the VIP manual, or back issues of the VIPER for a complete hexadecimal listing of the OS and the CHIP-8 interpreter. See Table 1 for a summary of instructions that must be changed in the OS to be compatible with the ELF-II. Once you have loaded the OS, with the modifications, put it on tape before

you run it, in case you make a mistake while loading it.

ADDRESS	CODE	COMMENTS
8001	0D	;CHANGES PAGE LOCATION OF CODE
800A	67	;OUTPUT PORT ENABLE FOR KEYPAD
8022	37	;EF4 INSTEAD OF EF3
8056	0E	;CHANGES PAGE LOCATION OF CODE
8104	3F	;EF4 INSTEAD OF EF3
819C	67	;OUTPUT PORT ENABLE FOR KEYPAD
819F	3F	;EF4 INSTEAD OF EF3
81AA	37	;EF4 INSTEAD OF EF3

If you want to use EF3 for your keyboard, don't change the values at locations 8022, 8104, 819F or 81AA. High addresses 80 and 81 will become 0D and 0E when you load the code into RAM, starting at address 00D0.

TABLE 1 - OPERATING SYSTEM MODIFICATIONS

Now you can try the operating system by inserting a long branch (C0 0D 00) at address 0000. When you flip to RUN, while keeping input depressed if you are using EF4, you will see some random bit pattern on the top of the screen and an operating stack toward the bottom of the screen when you release the input switch. Press INPUT four times, and and you should see some address appear at the bottom left, and the contents of that address at the bottom right. (All this assumes you have either not made the keyboard mods or that you have made the mods and selected EF4 as your keyboard flag.) If you have made the keyboard modifications, follow the instructions in the VIP manual for using the OS. If you haven't made the modifications, this is as far as you can go with the operating system, although you can still use CHIP-8 and some of the VIP games.

Once you're sure the OS is working correctly, you can then modify and load the CHIP-8 interpreter at addresses 0000 through 01FF. Table 2 contains the modifications that must be made to the CHIP-8 interpreter before using it on your ELF-II.

ADDRESS	CODE	
0000	00 00 00 00 00 00	
000A	0E	
010B	0E	Locations 0000-0002 will now be
012A	0E	used for a long branch to the
019A	67	CHIP-8 patch at 0CF0, VIP OS at
019E	37	0D00, or to the ELF II monitor
01A1	3F	at F000. Locations 0003-0005 are not used.

TABLE 2 - CHIP-8 INTERPRETER MODIFICATIONS

Because the CHIP-8 interpreter normally expects R1.1 to be initialized to the display page when RUN is flipped on, we have to add an additional patch at 0CF0 (or anywhere you like - see Table 3). This patch must be the first code executed when you run a CHIP-8 program. The original code in the CHIP-8 interpreter at addresses 0000-0005 is not used.

ADDRESS	CODE	
0CF0	F807	;Put 07 in RB.1 for CHIP-8 display page
0CF2	BB	
0CF3	FF01	;Subtract 1 for CHIP-8 variable storage area
0CF5	B2B6	;Put 06 in R2.1 and R6.1
0CF7	C000	;Long branch back to address 0006 to start
0CF9	06	

Note: There is no easy way to use the same display page for both the OS and for CHIP-8, so now the OS uses page F and CHIP-8 uses page 7. Using this method, you can display instructions for using the operating system in page F and won't destroy it when you run a CHIP-8 program. Enter the code from Table 5 if you want the instructions displayed on the screen when you bring up the OS. Your name or other info can be inserted at address 0F10 through 0F37 if desired.

TABLE 3 - CHIP-8 PATCH

As you have seen by now, the modifications are very minor. We've only changed the I/O instruction code, page address code, and EF flag instructions. You can experiment and change the software to suit your individual purposes. Refer to table 4 for a memory map of your VIP/ELF-II code. You will probably want to relocate some code if you have very large CHIP-8 programs to run. All programs written in CHIP-8 with no machine language subroutines should work on your ELF-II. You will probably have to modify VIP machine language programs to run on the ELF-II because of the VIP executing some code in the OS before executing a program at address 0000. At first glance, it may look like the VIP executed machine language programs beginning at 0000, when it really goes to address 8000 and initializes R1.1 before jumping to 0000. Keep this in mind and you should have no problems.

In general, you may now run any CHIP-8 program that only requires a single key depression without making the keyboard modifications. If you have made the keyboard mods, you're ready to run any VIP game written in the CHIP-8 language.

PAGE	USE
0	CHIP-8 interpreter
1	"
2	CHIP-8 programs
3	"
4	"
5	"
6	CHIP-8 work area (06A0 - 06FF)
7	CHIP-8 display page
8	Available
9	"
A	"
B	"
C	CHIP-8 patch (0CF0 - 0CF9)
D	VIP operating system
E	"
F	VIP OS display page

TABLE 4 - MEMORY MAP

1. RCA COSMAC VIP instruction manual (VIP-300)
2. RCA VIP Users Guide (VIP-320)
3. VIPER (issues 2 and 3, volume 1)
4. CHIP-8 AND VIP are trademarks of RCA
5. GIANT BOARD and ELF-II are trademarks of Netronics R & D Ltd.

ADDRESS	DATA
0F00	FF FF FF FF FF FF FF FF
0F10	83 C8 9C F0 00 00 00 00
0F18	82 08 88 80 00 00 00 00
0F20	83 8A 38 F0 00 00 00 00
0F28	82 0F 38 10 00 00 00 00
0F30	F3 CD 9C F0 00 00 00 00
0F38	00 00 00 00 00 00 00 00
0F40	D9 E0 0F 02 6C 88 03 C0
0F48	F9 20 09 04 7C 88 02 40
0F50	A9 E5 4F 08 54 A9 52 40
0F58	89 40 09 10 44 F8 02 40
0F60	89 20 09 20 44 D8 03 C0
0F68	00 00 00 00 00 00 00 00
0F70	F9 E0 0F 02 7C 88 03 C0
0F78	21 20 09 04 10 88 02 00
0F80	21 E5 4E 08 10 A9 53 80
0F88	21 40 09 10 10 F8 02 00
0F90	21 20 0F 20 10 D8 02 00

TABLE 5 - OPERATING SYSTEM INSTRUCTION DISPLAY AREA

VIP HARDWARE AND SOFTWARE

The prices here are believed to be accurate and effective as of June 1, 1979. You may order RCA produced items from ARESCO, as well as the VIP software. Be sure to write on you order for any RCA products that you are aware that delivery may take more than 30 days.

RCA PRODUCED PRODUCTS

VP44	Four 2114 RAMs for on-board expansion	$ 36.00
VP45	Four 9131 RAMs for on-board expansion	36.00
VP311	VIP Instruction Manual	5.00
VP320	VIP User Guide	5.00
VP550	Supersound Board	49.00
VP560	EPROM Board	34.00
VP565	EPROM Programmer Board	99.00
VP570	Memory Expansion (4K static RAM)	95.00
VP575	System Expansion	59.00
VP580	Expansion Keypad	15.00
VP585	Keypad Interface Board	10.00
VP590	Color Board	69.00
VP595	Simple Sound Board	24.00
VP600	ASCII Keyboard (available 2nd quarter)	49.00
VP700	Tiny BASIC ROM Board	39.00
VP710	VIP Game Manual (20 games)	10.00
VP711	VIP (Assembled)	249.00
TC1210	9" Video Monitor	195.00
TC1212	12" Video Monitor	325.00
TC1217	17" Video Monitor	435.00
CDP18S731	Four 9131 RAMs plus necessary components for I/O expansion ports. Used only with the kits.	69.00
CDP18S745	Four 2114 RAMs plus necessary components for I/O expansion ports. Used only with the kits.	69.00
MPM201B	CDP 1802 User Manual	5.00

ARESCO SOFTWARE

Sam Hersh Editor.(cassette only)	5.00
Don Stein Editor (with documentation)	15.00
Don Stein Editor (cassette only)	5.00
Brian Astle's LIFE (with documentation)	10.00
Clarke Hottle's BASEBALL (cassette only)	10.00
Udo Pernisz's LUNAR LANDER (cassette only)	5.00
Robert Rupp's BLACKJACK.(cassette only)	5.00
Stanley Bayless's TREASURE HUNT (cassette only)	5.00
Robert Rupp's CRAPS (cassette only)	5.00
Tom Swan's PIPS FOR VIPS (book and cassette until 7/15/79)	14.95
Tom Swan's PIPS FOR VIPS (book and cassette after 7/15/79)	19.95

The book will be sold without the cassette, for 14.95, but the cassette will not be sold without the book.

We also have the 1861 data sheet (for $1.50). Please send an additional $1.00 for postage and handling if you order an RCA item. RCA charges us postage from their plant to our house, and we prepay shipping from our house to yours. Your dollar will help defray the costs of all that!

VIPER

The only comprehensive monthly newsletter devoted exclusively to

THE RCA COSMAC VIP

FEATURES:

* Expanded Explanation Of CHIP-8
* Interfacing an ASCII Keyboard To Your VIP
* A Text Editor — Or Two — Or Three — For The VIP
* Disassembled CHIP-8 PLUS Flowcharts
* I/O Mods For CHIP-8
* Double Resolution Graphics With CHIP-8
* Advance Notice: RCA's Newest VIP Peripheral Products
* Breakpoint Debug System
* Disassembled ROM Listing PLUS Annotation & Remarks
* LED Display Monitor

These are only some of the articles which appeared in the first four (that's right - all this - and more - in only four) issues of THE VIPER!

A year's subscription to THE VIPER includes all ten issues (every month except June & December) of the current Volume - so even if you're late getting your subcription order in, you won't miss any of the exciting features already published. And there's even a column for you - to share your favorite VIP project, and to read about other VIPper's experiences.

Send your check for $15.00 (or use your VISA, MC, or BAC card) for a full year's subscription. Outside USA and Canada, add $10 for overseas airmail. Sorry — no C.O.D. or billing is available. Mail your order to THE VIPER, P.O. Box 1142, Columbia, MD 21044.

NAME _____

ADDRESS _____
(Please give street address, not postoffice box, for UPS shipping)

CITY, STATE, ZIP CODE _____

MC/VISA/BAC No. _____ BANK No. (MC ONLY) _____

EXPIRATION DATE _____ SIGNATURE _____
(Required with credit card orders)

☐ Vol. 1 (July 1978 - May 1979) ☐ Vol. 2 (July 1979 - May 1980)

THE VIPER * P.O. Box 1142 * Columbia, * MD * 21044 * (301) 730-5186