

VIP2K rev.B1 Assembly and Operation

(Not a complete manual yet; but we're working on it!)

TMSI ElectroniKits

814 8th Ave N

Sartell MN 56377

leeahart@earthlink.net

<http://www.sunrise-ev.com/vip2k.htm>

Last update: 25 Oct 2021

Before the Apple, Atari, and Commodore home computers, Joe Weisbecker created the [RCA VIP](#). Introduced in 1976, it was a simple, elegant, and low cost design so that everyone could have fun and learn about personal computers. It had an RCA 1802 microprocessor, 4k of RAM, a monitor program in ROM, a 16-key hex keypad, a 64x128 pixel graphics video display, and a serial port to load/save its programs on cassette tapes. That's an original RCA VIP on top of the monitor in the photo below.

For the 40th anniversary of the VIP, we decided to celebrate by making a new version you can build yourself. That's it at the bottom of the photo, below the monitor, with its tiny keyboard on top! The black box at the right is just the battery box, with four AA cells to power it.

The VIP2K has the same 1802 microprocessor, but with several significant upgrades in speed, memory, and features:

- 1802 microprocessor running at 4 MHz
- 32K of RAM
- 32K of ROM, with Monitor, BASIC, and CHIP-8
- NTSC or PAL video output displays
 - 24 lines of 24 text characters
 - 192 x 192 pixel graphics
- 44-key full ASCII keyboard
- TTL serial I/O port up to 9600 baud
- built entirely with vintage parts and through-hole technology
- ...and it all fits in a 3.5" x 2" x 0.75" Altoids tin!

This project is under development, so things are still changing. This manual is a "snapshot" of where we are today.

Check the website (at the link on the top of the page) for details on operation, and the latest software developments.



Credits: This project would not have been possible without the inspiration of Joseph Weisbecker, a true microcomputer pioneer. Thanks also to Lee Hart for the hardware design, Chuck Yakym for adapting his Monitor and RCA's BASIC3, and Marcel van Tongeren for CHIP-8 and his EMMA 1802 emulator.

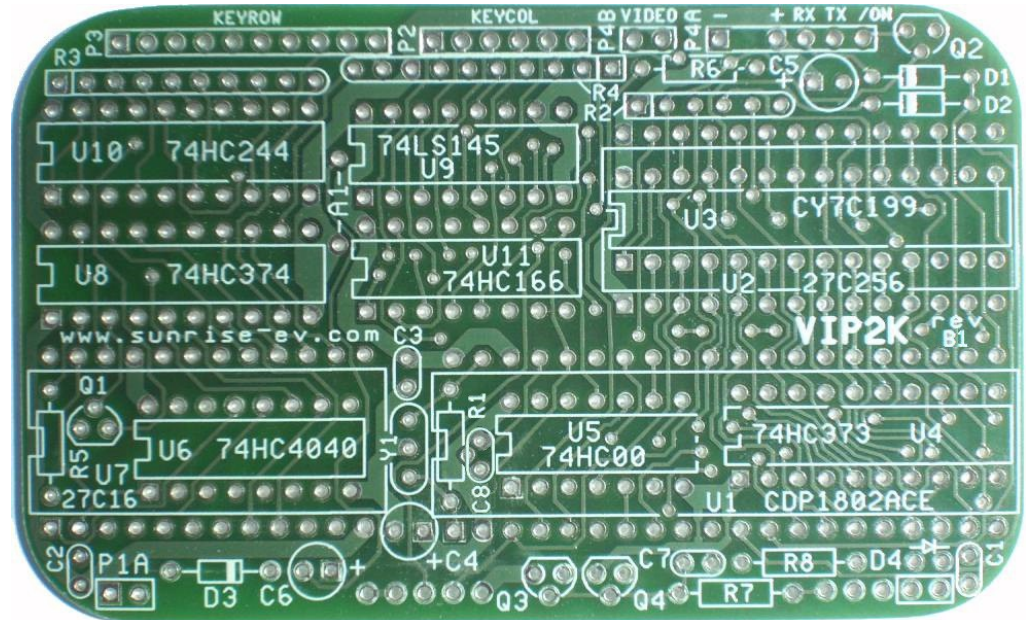
Parts List

<u>Qty</u>	<u>Reference</u>	<u>Description</u>	<u>Source</u>
3	C1, C2, C3	capacitor 0.1uF 50v X7R 0.1"LS	Jameco.com 1570161
3	C4, C5, C6	capacitor 4.7uF 10vdc tantalum	Jameco.com 94035
1	C7	capacitor 0.01uF 50v X7R 0.1"LS	Jameco.com 544833
1	C8	capacitor 3300pF ceramic	Mouser.com 594-K332K15X7RF5TL2
3	D1, D2, D3	diode 1N4148	Jameco.com 36038
1	D4	red LED, T1, flat top, without flange	Mouser.com 696-SSL-LX30FT4ID
1	P1a,b,P1-P4	36-pin male pin header: Cut it to make...	Jameco.com 68339
2		P1a, P1b = 2-pin header (CPU J1 pins 1-2 and 29-30),	
1		P1 = 2-pin header VIDEO (CPU card),	
1		P2 = 6-pin header KEYROW (CPU card),	
1		P3 = 10-pin header KEYCOL (CPU card),	
1		P4 = 6-pin header POWER+SER (CPU card).	
2	J1a,b, J2	6-pin male/female stacking header	Jameco.com 2144614
2	J3, J4	10-pin male/female stacking header	Jameco.com 2177627
1	Q1	FJN4303 PNP transistor w. base resistors	Arrow FJN4303RTA
1	Q2	FJN3303 NPN transistor w. base resistors	Digikey 2368-NTE2358-ND
1	Q3	2N3904 NPN transistor	Jameco.com 38359
1	Q4	2N3906 PNP transistor	Jameco.com 38375
1	R1	10Meg 5% 1/4w resistor	Jameco.com 691817
1	R2	10K x 5 6-pin SIP bussed	Mouser.com 652-4606X-1LF-10K
1	R3	10K x 9 10-pin SIP bussed	Mouser.com 652-4610X-1LF-10K
1	R4	820 x 9 (or 1K x 9) 10-pin SIP bussed	Mouser.com 652-4610X-1LF-1K
1	R5	100K 5% 1/4w resistor	Jameco.com 691340
1	R6	270 ohm 5% (or 267 ohm) 1/4w resistor	Jameco.com 690726
1	R7	1K 5% 1/4w resistor	Jameco.com 690865
1	R8	2.0K 5% 1/4w resistor	Jameco.com 690937
44	S1-S44	tactile switch, Alps SKHHAJA010 or eq.	Mouser.com 688-SKHHAJ
1	U1	CDP1802ACE microprocessor (NOTE 1)	ebay etc.
2	U1s	20-pin socket strip (to socket U1)	Jameco.com 41136
1	U2	27C256 32K EPROM with VIP2K15.hex	Jameco.com 39714 (need to program it)
2	U2s	14-pin socket strip (to socket U2)	Jameco.com 40328
1	U3	CY7C199 (or equivalent) 32k RAM	Jameco.com 242376
1	U4	74HC373 octal transparent latch	Jameco.com 45831
1	U5	74HC00 quad 2-input NAND gate	Jameco.com 45161
1	U6	74HC4040 12-bit binary counter	Jameco.com 45920
1	U7	27C16 2K EPROM with 2716NTSC or 2716-PAL	Jameco.com 40011 (need to program it)
2	U7s	12-pin socket strip (to socket U7)	Jameco.com 39351
1	U8	74HC374 octal latch	Jameco.com 45858
1	U9	74LS145 (or 74HC145) BCD decoder	Jameco.com 46666
1	U10	74HC244 octal buffer	Jameco.com 45655
1	U11	74HC166 8-bit shift register	Mouser.com 595-SN74HC166N
1	Y1	resonator 4MHz with capacitors	Mouser.com 81-CSTS0400MG03
1	A1	speaker	BGmicro.com AUD1162
1	VIP2K	PC card & keyboard, rev.B1	www.sunrise-ev.com/vip2k.htm

NOTE 1: The VIP2K runs the 1802 at 4 MHz. The original non-A CDP1802 was specified to 2.5 MHz, and 3.2 MHz for the later CDP1802A. This means you have to **select** a fast enough 1802. Some non-A 1802's are too slow; but most 1802A's easily run at 4 MHz; that's what you want.

To select a fast enough part, look at the waveform on 1802 pin 39 (/XTAL out) with an oscilloscope and 10x probe. It should be a 4MHz sine wave, 4 volts AC peak-to-peak, going from about 0.5v to 4.5v (with VCC=5v). 4vpp is needed to clock video shift register U11. Kits include a tested 4 MHz 1802A.

VIP2K Card Assembly



Assembly is a work in progress; so I'll just tell you how I built mine. Let me know if you find a better way! Most parts are easy to install (just like any other kit), but I added notes for the “tricky” parts:

- () Install resistors R1, R5, R6, R7, and R8.
- () Install 0.1uF capacitors C1, C2, and C3 (marked "104").
- () Install 0.01uF capacitor C7 (marked "103").
- () Install diodes D1, D2, and D3. The end with the band must match the board.
- () Install U3, U4, U5, U6, U8, U9, U10, U11. Sockets are not supplied, but you can add them if you like. Note that U4, U5, and U6 are **under** ICs! If you want to socket them, I recommend **socket pins** (digikey.com ED5037-ND). They drop right into the holes to make a “zero height” socket.
- () Install transistor Q1 (marked “R4303”). Q1 fits **under** U7. Put its wires in the holes, with the flat side as shown. Then bend it over so the flat side is tight against the board. **Now** solder it in.
- () Install IC sockets for U1, U2, and U7. Use socket strips (supplied) or IC sockets. If sockets, cut out the thin plastic bars between the left and right sides to make room for the parts underneath.
- () Install SIP resistors R2 (black), R3 (black), and R4 (red). Put pin 1 on the **left** end for R2 and R3; and the **right** end for R4. Don't mix up R3 and R4 – they are **different** values!
- () Install transistor Q2 (marked “R3303”), Q3 (marked "2N3904"), and Q4 (marked "2N3906").
- () Install ceramic resonator Y1.
- () Install capacitors C4, C5, and C6. They are **polarized**; the + end must match the board.
- () Install 3300pF capacitor C8 (marked "332").

Headers: I supplied a 36-pin header. Cut it up to make P1a, P1b, and P1-P4:

- () P1a and P1b: Install a 2-pin header at each lower corner of the card. P1a is on the left, and P1b on the right (but there was no room to label P1a). They go on top of the board, in the holes **closest to the outer edge** of the board. They serve as “feet” to support the Keyboard.
- () Install P2, a 6-pin header on the VIP2K card at the KEYCOL location.
- () Install P3, a 10-pin header on VIP2K card at the KEYROW location.
- () Install P4, a 9-pin header on the VIP2K card at the locations labeled VIDEO – + RX TX /ON. Remove the 3rd and 5th pins for keying (there are no holes for them on the card).

- () Install a 2716 or 27C16 EPROM, or 28C16 EEPROM at U7. Program and label it with:
 - NTSC (60Hz): Use the file at <http://sunrise-ev.com/photos/1802/2716ntsc.hex>
 - PAL (50Hz): Use the file at <http://sunrise-ev.com/photos/1802/2716-pal.hex>
- () Install a 27256 or 27C256 EPROM programmed with VIP2K15.HEX at U7. This file can be downloaded at <http://sunrise-ev.com/photos/1802/vip2k15.hex>
- () Install CDP1802ACE microprocessor U1. It must work at 4 MHz; see **Note 1** on page 2.
- () Install speaker A1 on top of U8 and U10. Connect it to the holes labeled "A1" with scrap wire. The black side with the "W" faces up.

Keyboard Assembly

The male/female stacking connectors (as used on Arduinos etc.) go on the **bottom** of the Keyboard. To be sure the connectors line up, plug the female part onto the pins on the VIP2K board. Put the Keyboard on top. Then solder the pins on top of the keyboard. Cut off the excess pin length.

- () Install J2, a 6-pin stacking connector. Plug the female part onto KEYCOL on the VIP2K card.
- () Install J3, a 10-pin stacking connector. Plug the female part onto KEYROW on the VIP2K card.
- () Install J1a and J1b. I bought 6-pin connectors, and cut one down to make 2-pin connectors for J1a and J1b. To cut it, pull out a pin with pliers, then cut the plastic body in the center of the removed pin location with a sharp knife or diagonal cutters. Plug these onto P1a and P1b.
- () Install J1+J4. I used a single 10-pin part, and removed pins 7 and 10. Plug it onto P1 (VIDEO) and P4 (POWER/SERIAL) on the VIP2K card. The male pins are long, and not soldered to the Keyboard. Use them to connect to your video monitor, power, and serial I/O.
- () Install red LED D4 on the VIP board. The long wire goes in the left arrowhead (anode) hole. Plug the keyboard on top to hold the LED in the hole in the keyboard. Then solder the leads.
- () Install tactile switches S1-S44. Cut off the excess pin length on the back to prevent shorts.

Testing

Connect a video monitor to P1, and 5v power to P4 + and -. You should see the Starship Enterprise for a moment, then the Monitor sign-on message. If not, try another power-on reset (CTL and RES keys). Supply current is ~25ma with CMOS EPROMs (supplied), or ~80ma with 27xx NMOS EPROMs.

In case of difficulty, check for a 4vpp 4MHz signal on U1 pin 39 (see **Note 1** on page 2). The 1802 is cleared on power-up, then runs the program in U2. Look for pulses on TPA, TPB and SC0 (U1 pins 34, 33, and 6). U2 has a video interrupt and DMA handler; so check for pulses on /INT and /DMA-OUT (U1 pin 36 and 37).

U6-U7-U8 generate video independently from the 1802. U8 pins 2-5 divide TPA by 4 to clock U6. U7 decodes the count from U6 to produce SYNC, BLANK, /DMA, and /INT signals on U8.

DMA is enabled by the TVON latch (Q1-R5-U8 bit 0). An IN6 instruction turns it ON, and IN7 turns it off. The ROM program initially turns TV OFF until R0 and the interrupt handler are initialized. If you are having problems debugging, ground U9 pin 9 to set TV ON=0 to disable unwanted DMAs and interrupts.



"OK, I think I found your problem. You didn't use the right swear words to assemble it."

Operation

Now the fun begins! :-) The monitor displays a "Starship Enterprise" image, then signs on like this:

```
VIP2K Monitor Ver. 1.5
Enter "H" for Help.
>_
```

The >_ prompt means the Monitor is waiting for a command letter. Commands must be UPPERCASE. Some (like H for Help or B for BASIC) act immediately. Others are followed by one or more hex numbers (0-9 A-F), separated by a SP (space) key, then a final CR key (also called Enter or Return). Leading zeros are not needed (so F8 is the same as 00F8). Backspace is not used; if you make a mistake entering a number, continue to enter the corrected number and only the last 2 or 4 digits will be used. To abort a command, type <ESC> (control-1; press and hold the CTL key, then press the digit 1 key).

H Help

Type H for a Help screen with a brief command summary. All Monitor and BASIC commands are CAPITAL letters, so the keyboard input routine defaults to upper-case. To enter lower-case letters (for example, in BASIC PRINT statements), press and hold down either SH (Shift) key, then enter the letter.

```
Enter Command letter
to see help text.
To exit HELP, Press ESC
```

```
Commands are:
M reads memory
W writes memory
T transfer memory
R run program
U view 1802 registers
D disassembler
B BASIC 3 V1.1
S save program
L load program
C Run CHIP 8
I loads CHIP 8 program
A About
```

```
?_
```

The ?_ prompt tells you the HELP command is active. Type a key for additional help about that command. For example, M will display more information about the Read Memory command. When finished with Help, type <ESC> (CTL-1) to exit.

M Read Memory

Maaaa bbbb <CR>

Reads **bbbb** bytes of memory, starting at address **aaaa**. For example, to read 7 hex bytes starting at address 0100, type M100 7<CR>.

```
>M100 7
0100 A7 F8 FF 57
0104 C0 0D 00
>_
```

The Read Memory command displays the address, then 4 bytes of data, and continues until 7 bytes have been displayed, or until you type <ESC> CTL-1 to abort.

W Write Memory

Waaaa dd dd ... <CR>

Writes one or more data bytes **dd** into memory, starting at address **aaaa**, until the final <CR> (the CR key). Separate each byte with the <space> SP key.

```
>W9000 00 11 22 33 44 55
   66 77 88 99 AA BB CC DD
   D0EE FF
>_
```

The Write Memory command writes bytes to RAM. The screen scrolls as needed for long entries. Oops, DD is an error! Type EE to correct it. End with <CR> CR, or <ESC> CTL-1 to abort.

T Transfer Memory

Taaaa bbbb cccc <CR>

Transfer (block-copy) **cccc** bytes from address **aaaa** to **bbbb**. All three numbers are **required**. Separate each number with a <space> SP. The transfer starts at the low-address end, copies a byte, increments both addresses, and repeats for cccc bytes. You can move overlapping blocks down. But if you move overlapping blocks up, the aaaa block data gets repeated. This is a way to initialize memory to a known value: For example, T9000 9001 100 will write the byte at 9000 into **every** byte from 9001 to 9101.

R Run Program

Raaaa <CR>

Run Program at address **aaaa**, with P=R3 and X=R2. R3 is the Program Counter, which is set to aaaa. R2 is the Stack Pointer, which is pointing to free RAM memory. All other registers are set to the values shown by the U (View Registers) command.

To return to the Monitor, your program should jump to 0D1B (C0 0D 1B in hex; or LBR 0D1Bh in assembler).

V View 1802 Registers

Raaaa <CR>

The View command shows the value of all 1802 registers that were saved when the Monitor was last entered. They will be random values on power-up, or whatever values were left there by the the last program running when it jumped to the Monitor.

```
>U
1802 Register Contents
R0 = 008E   R1 = 0039
R2 = E7E1   R3 = 4090

R4 = FFFF   R5 = 2FF2
R6 = 81A2   R7 = FEF9

R8 = 0264   R9 = E7CE
RA = 81A1   RB = 81DF

RC = 225B   RD = 3B02
RE = 010D   RF = 9701

RC = 225B   RD = 3B02
RE = 010D   RF = 9701

PC = F      X = F
T = FF      D = FF

DF = 1
>_
```

View registers (saved in these RAM at addresses):

register	hi-addr-lo	register	hi-addr-lo
R0	(E7DC-DD)	R1	(E7DE-DF)
R2	(E7E0-E1)	R3	(E7E2-E3)
R4	(E7E4-E5)	R5	(E7E6-E7)
R6	(E7E8-E9)	R7	(E7EA-EB)
R8	(E7EC-ED)	R9	(E7EE-EF)
RA	(E7F0-F1)	RB	(E7F2-F3)
RC	(E7F4-F5)	RD	(E7F6-F7)
RE	(E7F8-F9)	RF	(E7FA-FB)
RC	(E7F4-F5)	RD	(E7F6-F7)
RE	(E7F8-F9)	RF	(E7FA-FB)
P	(E7FC)	X	(E7FD)
T	(E7FE)	D	(E7FF)

The View command actually shows the register values saved in RAM. You can change the saved register values with the **W** (Write) command, using the addresses shown above. These new values will then be loaded into the 1802 registers when you use the **R** (Run) command.

Precautions: Only change the values of registers R0-R5 with great care! They are being used by the Monitor, so if you modify them, you may not be able to return to the monitor without a cold reset!

- R0 is used as the DMA pointer for the video display.
- R1 points to the interrupt handler, which is scanning the keyboard and display.
- R2 is the stack pointer.
- R3 is your program counter for the RUN command.
- R4 points to the SCRT "Call" subroutine
- R5 points to the SCRT "Return" subroutine

D Disassemble 1802 Opcodes Daaaa bbbb <CR>

Disassemble 1802 opcodes from starting address **aaaa** to ending address **bbbb**. Each line displays an address, the opcode at that address, and its assembler mnemonic, along with any data bytes the instruction needs. Example:

<pre>>DC9 CE 00C9 F8E8 LDI E8 00CB B7 PHI R7 00CC F800 LDI 00 00CE A7 PLO R7 >_</pre>	<p>Disassemble memory from 00C9 to 00CE: Load Immediate E8 into D ...and Put it in the High byte of Register 7 Load Immediate 00 into D ...and Put it in the Low byte of Register 7 Abort a long disassembly with <ESC> CTL-1</p>
---	--

If the ending address **bbbb** is 0 or equal to or less than the starting address, the Disassembler enters Single-Step mode. It will disassemble one instruction, and then wait. Press <CR> (the CR key) to continue, or <ESC> (CTL-1) to abort.

<pre>>DC9 0 00C9 F8E8 LDI E8_ 00CB B7 PHI R7_ 00CC F800 LDI 00_ Function Aborted >_</pre>	<p>Disassemble from 00C9 in Single-Step mode: Load Immediate E8 into D (press CR to continue) ...and Put it in the High byte of R7 (CR to continue) Load Immediate 00 into D Press <ESC> CTL-1 to abort</p>
--	---

For fun, you can disassemble the entire 1802 opcode-mnemonic table with the command `D7700 7826`.

B run BASIC3 B <CR>

The **B** command starts RCA's floating-point BASIC3 interpreter. It will display its sign-on message, and ask you whether you want to do a Cold or Warm start.

<pre>>B WELCOME TO THE 1802 RCA BASIC3 V1.1 (C)1981 RCA C/W? _</pre>	<p>Start BASIC3. The sign-on message. Press C for a "Cold" start (the first time BASIC is started), or W for a "Warm" start (to re-start BASIC when you already have a program in memory).</p>
---	--

BASIC3 will respond with READY and its ":" prompt. BASIC is now running and ready for your commands and programs. Here is a quick example:

```

READY
:10 PRINT 355/113
:20 END
:RUN
3.1416
READY
:BYE

```

BASIC3 is now running.
 Enter a BASIC program (hint: the / sign is CTL-9).
 Note that BS (the Backspace key) now works. :-)
 Run your program.
 Here's the result (it's approximately Pi).
 Ready for more...
 BYE exits BASIC and returns to the Monitor.

Here is a brief summary of BASIC3 commands. For a complete list of commands and operating details, see the BASIC3 User Manual at <http://sunrise-ev.com/MembershipCard/BASIC3v1user.pdf>

RCA BASIC3: Keywords are UPPER CASE. LIST to view, NEW to erase, BYE to exit.

Commands:	Definitions:	Controls:	I/O:	Math: (n=number)
BYE	DEFINT FIXED	END	DMAPT (n)	ABS (n) LOG (n)
CLD	DEFUS LET	EXIT	EFn	ATN (n) MEM
CLS	DEG RAD	GOSUB	INP (0, port)	COS (n) QST
DISINT	DIM REM	RETURN	INPUT	EOD PI
EDIT		GOTO	OUT (0, port, n)	EOP RND (n)
ENINT	Data:	WAIT (n)	PEEK (n)	EXP (n) SGN (n)
FORMAT	DATA	IF	POKE (addr, n)	FNUM (n) SIN (n)
LIST	READ	THEN	PRINT PR	INT (n) SQR (n)
NEW	RESTORE	FOR	QST STQ	INUM (n)
RENUMBER		TO		MOD (n1, n2)
RUN	File:	STEP	Math operators:	
RUN+	PLOAD	NEXT	+ - * / ^	String: \$="string"
TRACE	PSAVE		Logical:	ASC (\$)
			AND OR XOR NOT	CHR\$ (n)
			Relational:	FVAL (\$)
			= > < <> >= <=	LEN (\$)
			Hex: #FF = 255	MID (\$, start, n)
			@FFFF = 65535	TAB (n)
Machine language: sets R3=addr, P=3, CALL (addr, n1, n2) X=2, R8=n1, RA=n2; return with SEP R5 v=USR (addr, n1, n2) same, but sets v=R8				

S Save Program

Saaaa bbbb <CR>

The S (Save) command outputs **bbbb** bytes of memory, starting at address **aaaa** to the serial port. The output is in Intel hex (I8HEX) format, which is an ASCII text file with a checksum on each line. A PC or other serial device can display, print, or store this file on its disks. The saved file can be loaded later with the L (Load) command.

The default serial data rate is 9600 baud, 8N1 (1 Start, 8 data, no parity, 1 stop). The baud rate is stored in address E7CE, and can be changed with the Monitor W (Write) command, or BASIC POKE command. Note: Entering the Monitor from BASIC will reset the serial port back to 9600 baud.

```

>ME7CE 1
E7CE 0D
>WE7CE 19
>WE7CE 35
>WE7CE 68
>_

```

Read the current baud rate.
 0D = 9600 baud (the default).
 19 = 4800 baud (change to 4800 baud)
 35 = 2400 baud (change to 2400 baud)
 68 = 1200 baud (change to 1200 baud)

Set up your PC serial port and Terminal program **before** starting the `SAVE` command, so it is ready and waiting for data. Then enter the `S` (Save) command. Note: `<ESC>` does not abort a `SAVE` command.

```
>S8000 400
Ready to SAVE Program

File Saved Successfully
>_
```

Save 400 bytes of memory starting at 8000.
This message is BRIEFLY displayed...
Then the screen goes blank while sending the data.
This message is displayed when done sending.

L Load Program

L `<CR>`

The L (Load) command receives an Intel hex (I8HEX) format file on the serial port, and loads it into memory. The file format is the same one produced by the `SAVE` command. Intel hex files includes the starting and ending addresses, so these do not need to be supplied.

Set up your PC serial port and terminal program **before** starting the `LOAD` command. Use the program's "send" or "ASCII upload" command to select the file to send. Then enter L to start the Load process. Note: `<ESC>` on the VIP2K will not abort during the `LOAD` command; but you can type `<ESC>` on your PC to abort loading.

```
>L
Ready to LOAD Program

File Loaded Successfully
>_
```

Load an Intel hex file...
This message is BRIEFLY displayed...
Then the screen goes blank while receiving data.
Tell your PC to start sending data.
This message is displayed when sending is finished.
(or an error message if unsuccessful).

The default data rate is 9600 baud, but it can be changed as described for the `S` (Save) command.

Your PC must **limit the speed** that it sends data. This is typically called "Pacing", "Character delay", or "Transmit speed limiting". You will have to experiment to see how much delay is needed. In Realterm for example, set "Delays" in the "Send" tab to 5 for 9600 baud, 10 for 4800 baud, 15 for 2400 baud, or 20 for 1200 baud. Note that many USB-serial adapters **defeat** the pacing your terminal program adds.

C run CHIP-8

C `<CR>`

CHIP-8 is a simple, easy-to-learn interpreter, like BASIC. While BASIC is optimized for text, CHIP-8 is optimized for graphics and games. It was written in 1976 by Joe Weisbecker for the original RCA VIP and its 1861 video chip, but has been adapted to run on many other systems. Marcel van Tongeren wrote this version for the VIP2K. See Wikipedia <https://en.wikipedia.org/wiki/CHIP-8> and the CHIP-8 tutorial at <https://github.com/JohnEarnest/Octo/blob/gh-pages/docs/BeginnersGuide.md> for more information.

Any VIP CHIP-8 program should run on the VIP2K, but with some speed differences due to the faster 1802 and higher screen resolution. Sound support is missing. Enhanced versions such as CHIP-8X, SCHIP, CHIP-10, Chip ETI600, Chip8 Hires, and CHIP-8 using 1802 subroutines are not supported.

A CHIP-8 program must be loaded by hand, or with the I command **before** you run CHIP-8 itself. If no CHIP-8 program loaded, the C command says `No CHIP-8 File Loaded` and returns to the Monitor. To load a CHIP-8 program manually, enter its hex opcodes with the `W` command at address 8200 hex:

```
>W8200 XX XX XX XX XX XX ...
>WFEFD 80
>C
```

Each opcode, with space between, `<CR>` to end.
Set the "CHIP-8 program loaded" flag.
Now use the C command to run CHIP-8.

I Input CHIP-8 program

I <CR>

The I (Input) command loads a CHIP-8 program in the same Intel Hex (I8HEX) format as the L (Load) command. The only difference between the two is that I (Input) also sets a flag to indicate that a CHIP-8 program has been loaded.

Sample CHIP-8 programs can be downloaded at <http://sunrise-ev.com/photos/1802/c8games.zip> This is a ZIP file containing a couple dozen games, which you will need to unzip. They are in Intel hex (I8HEX) format, already set for the correct load address set.

Choose a game, and set up your PC serial port and Terminal program to send it in "text" mode as described for the L (Load) command. Then load it with the I command...

```
>I
Ready to LOAD Program

File Loaded Successfully
>MFEFD 1
>FEFD 80
>C
```

Load a CHIP-8 Intel hex file...
This message is BRIEFLY displayed...
Then the screen goes blank while waiting for data.
Tell your PC to start sending data.
This message is displayed when it's been received.
Let's check the "CHIP-8 Loaded" flag at FEFD.
It will be 80 if successful (or 00 if unsuccessful).
NOW you can use the C command to run CHIP-8.

CHIP-8 Operation

Keyboard commands while CHIP-8 is running:

- CTL-0 Quit CHIP-8 and return to the Monitor. The CHIP-8 "Loaded" flag is set to 00
- CTL-1 to CTL-9 Set CHIP-8 speed (where 1 is the slowest, 9 is the fastest)
- CTL-K Reset keyboard map to the default
- CTL-R Reset CHIP-8 interpreter
- CTL-S Reset CHIP-8 interpreter in SMALL screen mode (2x4 pixels)
- CTL-L Reset CHIP-8 interpreter in LARGE screen mode (3x6 pixels)

The SMALL screen mode updates faster, and is recommended for programs with a "busy" screen.

Keyboard Map: The original VIP had a 4x4 keypad labeled 0-9, A-F. The VIP2K has the same keys, but in different positions. Many CHIP-8 programs ignored the key labels and simply used the key positions to select game functions (up / down / left / right / fire). So the VIP2K also maps I=up (2), J=left (6), K=right (6), M=down (8), and SP=fire (5) as used by VIP games.

		^		
1	2	3	C	
<	fire	>		
4	5	6	D	
	V			
7	8	9	E	
A	0	B	F	

RCA VIP keypad
red=game functions

slowest								fastest		
1	2	3	4	5	6	7	8	9	0	
quit			reset					^		
Q		E	R				I			
		small					<	>	large	
CTL	A	S	D	F			J	K	L	
								V		
			C		B			M		

VIP2K keyboard
running CHIP-8 | fire | red=game functions
| SP | blue=CTL functions

The key map table is in RAM at FF00-FFAF. It can be changed to suit each game, and loaded as part of the game Hex file itself. Here are some typical locations:

FF0A J key code
 FF0B M key code
 FF11 I key code
 FF12 K key code
 FF1B <SP> key code
 FFA8 game speed value
 FFA9 screen size

CHIP-8 Memory Map

8000-81FF Normally not used, but available for CHIP-8 code
 8200-8FFF CHIP-8 user space

E800-EC77 Video RAM in LARGE 3x5 pixel format:
 E800-E84D - Top 3 lines; not used by CHIP-8 (should always be 0)
 E84E-EB8D - CHIP-8 video screen
 EB8E-EBF5 - Bottom 4 lines; used for some games like Pong
 EBF6-EC77 - Bottom 5 lines; not used by CHIP-8 (should always be 0)

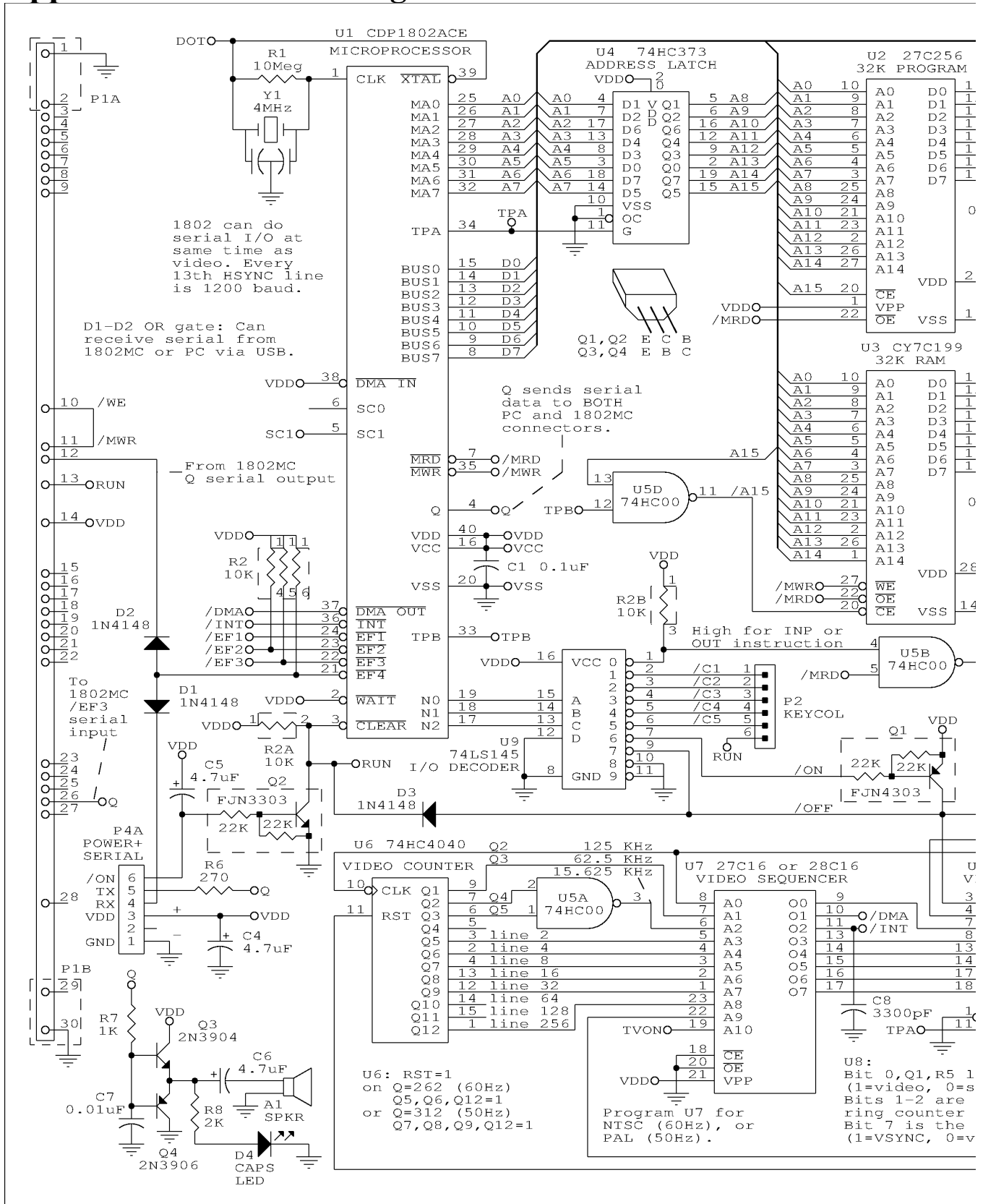
E800-ED95 Video RAM in SMALL 2x4 pixel format:
 E800-E8CF - Top 8 blank lines; not used by CHIP-8 (should always be 0)
 E8D0-EC0F - CHIP-8 video screen
 EC10-ECF9 - Bottom 9 lines; used for some games like Pong
 EBFA-ED95 - Bottom 15 blank lines; not used by CHIP-8 (should always be 0)

FEFD CHIP-8 Loaded flag (00 if not loaded, 80 if CHIP-8 program is loaded)
 FF00-FF9F Keyboard mapping table
 FFA0-FFA4 CHIP-8 identifier text ("CHIP8")
 FFA8 Speed, 0-30 hex, in steps of 6
 FFA9 Screen resolution; 0=large 3x5 pixels, not 0=small 2x4 pixels
 FFB2-FFCF Jump table for CHIP-8 instructions
 FFE0-FFEF CHIP-8 variables V0-VF
 FFF0-FFF3 Graphic scratchpad area
 FFF9 Keyboard code
 FFFA CHIP-8 counter. Counts down to 0 from value set by CHIP-9 instruction

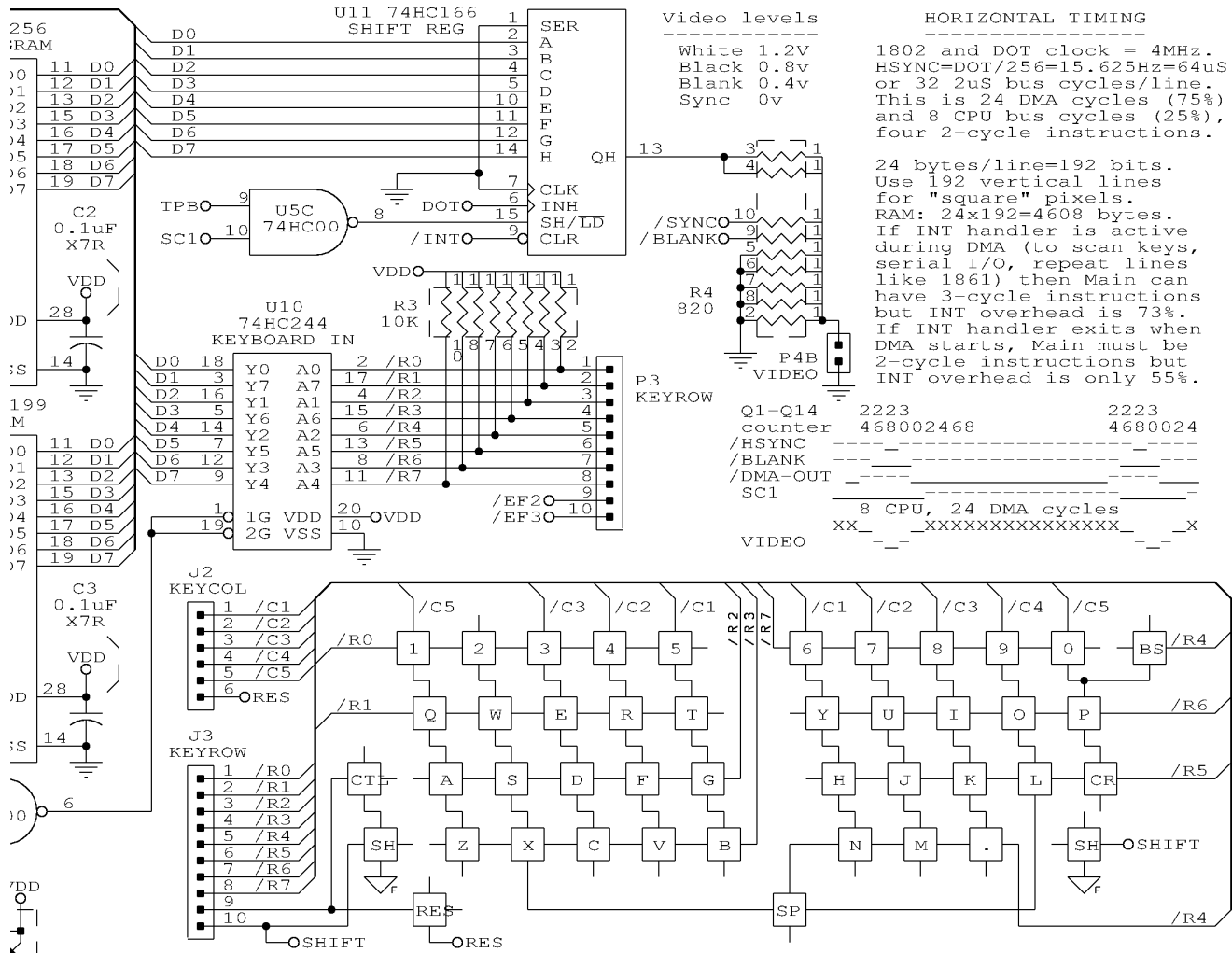
Differences in CHIP-8 instructions

	Original VIP CHIP-8	VIP2K CHIP-8
User Space	0200-0EFF	8000-8FFF (starting address is 8200)
00aa, SYS 0aa		Call 1802 system routine at 70aa
0aaa, SYS aaa	Call 1802 system routine at aaa	Call 1802 system routine at 8aaa (aaa>0FF)
1aaa, JP aaa	Jump to address aaa	Jump to address 8aaa
2aaa, CALL aaa	Call subroutine at aaa	Call subroutine at 8aaa
Aaaa, LD I,aaa	I = aaa	I = 8aaa
Baaa, JP V0,aaa	Jump to address aaa + V0	Jump to address 8aaa + V0
Fx18, LD ST,Vx	Sound timer = Vx	NOP

Appendix A -- Circuit Diagram



vip2kb-sch1.bmp



Video levels
 White 1.2V
 Black 0.8V
 Blank 0.4V
 Sync 0V

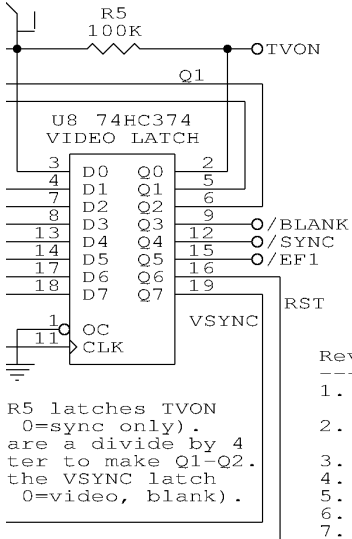
HORIZONTAL TIMING
 1802 and DOT clock = 4MHz.
 HSYNC-DOT/256=15.625Hz=64uS
 or 32 2uS bus cycles/line.
 This is 24 DMA cycles (75%)
 and 8 CPU bus cycles (25%),
 four 2-cycle instructions.

24 bytes/line=192 bits.
 Use 192 vertical lines
 for "square" pixels.
 RAM: 24x192=4608 bytes.
 If INT handler is active
 during DMA (to scan keys,
 serial I/O, repeat lines
 like 1861) then Main can
 have 3-cycle instructions
 but INT overhead is 73%.
 If INT handler exits when
 DMA starts, Main must be
 2-cycle instructions but
 INT overhead is only 55%.

Q1-Q14 counter	2223 468002468	2223 4680024
/HSYNC	-----	-----
/BLANK	-----	-----
/DMA-OUT	-----	-----
SC1	-----	-----
VIDEO	XX_ _XXXXXXXXXXXXXXXXXX_ _X	XX_ _XXXXXXXXXXXXXXXXXX_ _X

VERTICAL TIMING

NTSC: 262 lines, 59.637Hz. PAL: 312 lines, 50.08Hz.
 /VSYNC low on lines 0-11 (U6 Q12=0, Q7:Q6=00, 01, or 10).
 /INT low on lines 38-39 (Q12=0, Q9=1, Q6=1, Q5=1).
 Line 0-11: VSYNC, with 3 cycles of inverted HSYNC.
 Line 12-39: Blank (top margin).
 Line 38-39: /INT=0. 69 bus cycles before 1st DMA to save regs
 and initialize. End w. IDL to wait for DMA (allow 3-cycle inst.)
 Line 40-231: DMA (192 lines of 24 DMA cycles).
 Also, /EF1=0 tells Interrupt handler DMA is still in progress.
 Use to repeat lines to save RAM and reduce vertical resolution.
 Line 232-261 (or 232-311): Blank (bottom margin).
 Line 262 (or 312): RST=1 to reset U6.
 bottom margin-><-----vertical sync pulse-----><-top margin



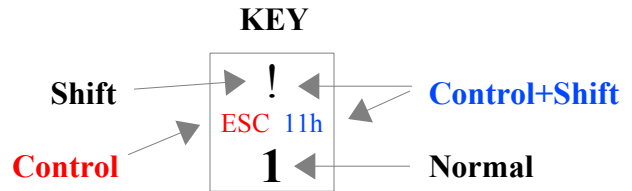
- Rev.B:
1. U11 was 74HC165, now 74HC166.
 2. Connect /INT to U11 /CLR (removes INT.ACK noise).
 3. Added Speaker output.
 4. Added CAPS lock LED.
 5. Improved Power-on Clear.
 6. CTL-RESET resets 1802.
 7. Add C8=3300pF from /INT to GND.

TMSI c/o Lee Hart		
814 8th Ave N		
Sartell MN 56377		
leeahart@earthlink.net		
Title		
1802 VIP2K - VIP in an Altoids tin		
Size	Document Number	REV
B	C:\ORCAD\SHEET\1802\VIP2KB.SCH	B1
Date:	March 1, 2021	Sheet 1 of 1

Appendix B -- Keyboard layout and Key Codes

! ESC 11h 1	@ NULL ; 1Ch 2	# ? 1Dh 3	\$: 1Fh 4	% + 1Eh 5	^ - 6	& * 7	* / 8	(= 9) = 0	Back-space BS
" Q	q ` 17h W	w ~ 05h HT 12h E	e HT 12h R	r 14h T	t 14h Y	y 19h VT 15h U	u 15h I	i HT O	o 0Fh P	p DEL 10h
control CTL	\ 01h A	_ 13h LF 04h S	d LF 04h D	f [06h F	g] 07h G	h STX BS H	j { LF J	k } VT K	l BS FF L	CR CapLock CR
Shift SH	z 1Ah Z	x 18h FF 03h X	c FF 03h C	v 16h V	b STX B	n 0Eh N	m < CR > M	> . .	Shift SH	
	RES				Space SP					

ASCII name	hex code	common function
NULL	00h	idle
STX	02h	
BS	08h	backspace
HTAB	09h	tab
LF	0Ah	line feed
VT	0Bh	vertical tab
FF	0Ch	form feed
CR	0Dh	carriage return
ESC	1Bh	escape
SP	20h	space
DEL	7Fh	delete



CTL + RES does a hardware RESET of the 1802.
CTL + CR toggles the CAPS lock mode (and its LED) on/off.
CTL + 1 is the ASCII "Escape" key.
 The SH (shift) and CTR (control) keys return no code on their own.
 Blank positions return a value of 00h (NULL).