VIP2K rev.A 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: July 6, 2019

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 the original 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 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
- 43-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 the Monitor and BASIC, and Marcel van Tongeren for CHIP-8 and his EMMA emulator.

Parts List

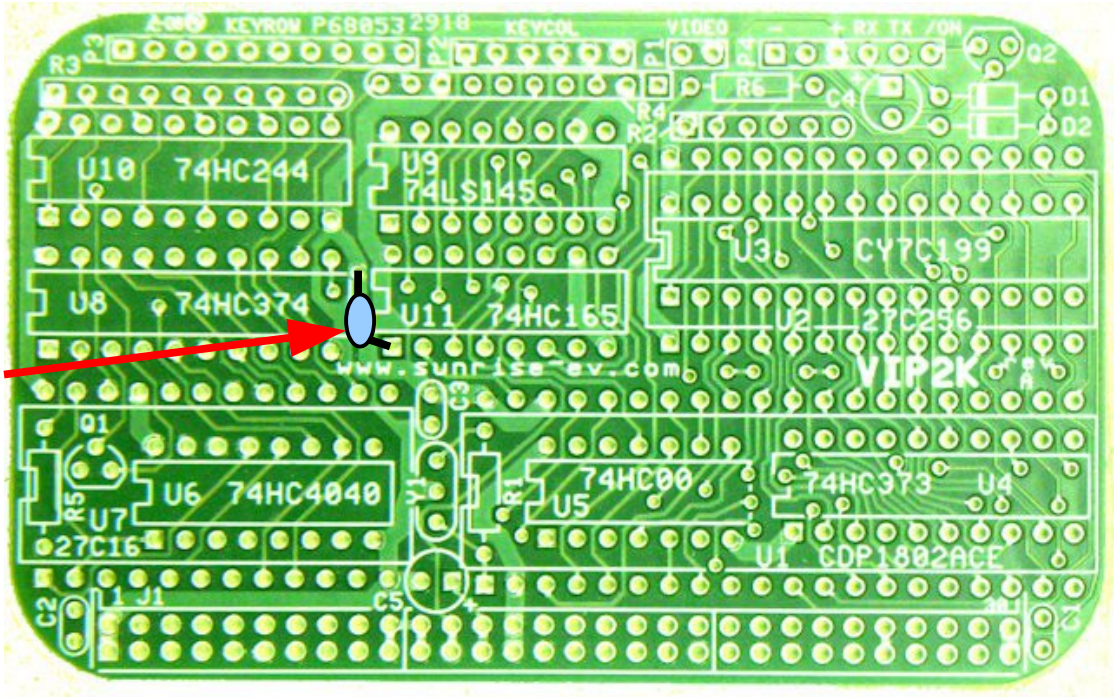
These are the parts supplied with the kit. If you bought the bare boards or are building a VIP2K from scratch, here are the parts you'll need:

Qty	Reference	Part	Source
3	C1, C2, C3	capacitor 0.1uF 50v X7R 0.1"LS	Jameco 1570161
2	C4, C5	capacitor 4.7uF 16vdc tantalum	Jameco 94035
1	C6	capacitor 3300pF ceramic	Mouser 594-K332K15X7RF5TL2
2	D1, D2	diode 1N4148	Jameco 36038
1	P1a,b,P1-P4	36-pin male pin header: Cut it to make	Jameco 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 2144614
2	J3, J4	10-pin male/female stacking header	Jameco 2177627
1	Q1	FJN4303 PNP transistor w. base resistors	Mouser 512-FJN4303RTA
1	Q2	FJN3303 NPN transistor w. base resistors	Mouser 512-FJN3303RTA
1	R1	10Meg 5% 1/4w resistor	Jameco 691817
1	R2	10K x 5 6-pin SIP bussed	Mouser 652-4606X-1LF-10K
1	R3	10K x 9 10-pin SIP bussed	Mouser 652-4610X-1LF-10K
1	R4	820 x 9 (or 1K x 9) 10-pin SIP bussed	Mouser 652-4610X-1LF-1K
1	R5	100K 5% 1/4w resistor	Jameco 691340
1	R6	270 ohm 5% (or 267 ohm) 1/4w resistor	Jameco 690726
43	S1-S43	tactile switch, Alps SKHHAJA010 or eq.	Mouser 688-SKHHAJ
1	U1	CDP1802ACE microprocessor (**NOTE 1**)	ebay etc.
1	U1s	40-pin socket strip (cut for U1 socket)	Jameco 41136
1	U2	27C256 32K EPROM with vip2k14.hex	Jameco 39714
1	U2s	28-pin socket strip (cut for U2 socket)	Jameco 40328
1	U3	CY7C199 (or equivalent) 32k RAM	Jameco 242376
1	U4	74HC373 octal transparent latch	Jameco 45831
1	U5	74HC00 quad 2-input NAND gate	Jameco 45161
1	U6	74HC4040 12-bit binary counter	Jameco 45920
1	U7	2716 2K EPROM with vip-2716.hex	Jameco 40011
2	U7s	12-pin socket strip (use for U7 socket)	Jameco 39351
1	U8	74HC374 octal latch	Jameco 45858
1	U9	74LS145 (or 74HC145) BCD decoder	Jameco 46666
1	U10	74HC244 octal buffer	Jameco 45655
1	U11	74HC165 8-bit shift register	Jameco 45495
1	Y1	resonator 4MHz with capacitors	Mouser 81-CSTS0400MG03
1	VIP2K	PC card & keyboard, rev.A	www.sunrise-ev.com/vip2k.htm

NOTE 1: The VIP2K 1802 runs at 4 MHz. The original non-A version CDP1802 was only specified to 2.5 MHz, or 3.2 MHz for the later CDP1802A. This means you have to **select** a fast enough 1802. I have found that most non-A 1802's are too slow. But later 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 on 1802 pin 16). 1802's supplied with the kits meet this criteria, and work in my VIP2K. A “slow” 1802 may oscillate at 4 MHz; but have too low a voltage to clock video shift register U11.

Assembly Notes



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, and R6.

( ) Install 0.1uF capacitors C1, C2, and C3.

( ) Install diodes D1 and D2. The end with the band must match the board.

( ) Install U3, U4, U5, U6, U8, U9, and U10. 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.

( ) U11: There is one cut-and-patch on the VIP2K rev.A card! **CUT** the trace on the bottom of the board between U11 pin 15-16. Then **ADD A WIRE** from U11 pin 15 to pin 8. Now install U11.

( ) Install transistor Q1 (marked "R4303"). Q1 must fit 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, R3, and R4. Put pin 1 on the **left** end for R2 and R3; **right** for R4. Don't mix up R3 and R4 – they look the same but are **different** values!

( ) Install transistor Q2. It is marked "R3303".

( ) Install ceramic resonator Y1.

( ) Install capacitors C4 and C5. They are polarized, so be sure to get the + and – right.

( ) Install C6. There is no place for it on the rev.A board. Put it between U8 and U11. Solder one end in the hole between U8 and U11 (GND). Solder the other end to U11 pin 1.

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 end of the 30-pin connector J1 on the VIP2K card. One in pins 1-2, and one in pins 29-30. They go on top of the board, in the holes **closest to the outer edge** of the board. These 2-pin headers just serve as "feet" to support the Keyboard.

( ) Install P1, a 2-pin header on the VIP2K card at the VIDEO location.

( ) 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 6-pin header on VIP2K card at the + / RX / TX / ON location.

( ) Install a 2716 or 27C16 EPROM, or 28C16 EEPROM at U7.
For NTSC video, use the file at http://sunrise-ev.com/photos/1802/2716ntsc.hex
For PAL video, use the file at http://sunrise-ev.com/photos/1802/2716-pal.hex

( ) Install a 27256 or 27C256 EPROM programmed with VIP2K14.HEX at U7. This file can be downloaded at http://sunrise-ev.com/photos/1802/vip2k14.hex

( ) Install CDP1802ACE microprocessor U1. It needs to be a 4 MHz part; see Note 1 on page 2.

Keyboard

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 tactile switches S1-S43. I have lots of SM (surface-mount) switches; so I used them. SM switch pins stick out horizontally. I pinch them with pliers to bend the pins down to fit into the holes on the Keyboard. Solder them from the TOP while pressing them tight against the board.

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 a few more power-on resets (sometimes it's fussy). Supply current is about 25ma with CMOS EPROMs supplied (or ~80ma with 27xx NMOS EPROMs).

In case of difficulty, check for a nice 4vpp 4MHz signal on 1802 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 (1802 pins 34, 33, and 6). The ROM program has a video Interrupt and DMA handler; so you should see pulses on /INT and /DMA-OUT (1802 pin 36 and 37).

Video sequencer U6-U7-U8 generates video independently from the 1802. It is clocked by TPA. The U6 outputs should count up. U7 converts the count into commands, to produce Hsync and Vsync pulses on U8 pins 9 and 12.

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 sign-on message looks like this:

```
VIP2K Monitor Ver. 1.4
Enter "H" for Help.
>_
```

The >_ prompt means the Monitor is waiting for a command letter. Some commands (like H for Help or B for BASIC) act immediately. Others are followed by one or more hexadecimal numbers (0-9 A-F), separated by a SP (space) key, then a final NL key (this is the ASCII <CR>, 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 (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 summary of the monitor commands. All Monitor and BASIC commands are capital letters, so the keyboard input routine defaults to upper-case. To enter lower-case letters, 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
V 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 NL key). Separate each byte with the <space> SP key.

```
>W9000 00 11 22 33 44 55
 66 77 88 99 AA BB CC DD
 DDEE 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> NL, 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. For example, T9000 9001 100 will write the byte at 9000 into **every** byte from 9001 to 9101. This is a way to initialize memory to a known value.

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 V (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.

```
>V
1802 Register Contents
R0 = 0D8E   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:

```
>DC9 CE
00C9 F8E8   LDI  E8
00CB B7     PHI  R7
00CC F800   LDI  00
00CE A7     PLO  R7
>_
```

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

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 NL key) to continue, or <ESC> (CTL-1) to abort.

```
>DC9 0
00C9 F8E8   LDI  E8_
00CB B7     PHI  R7_
00CC F800   LDI  00_
Function Aborted
>_
```

Disassemble from 00C9 in Single-Step mode:
Load Immediate E8 into D (press NL to continue)
...and Put it in the High byte of R7 (NL to continue)
Load Immediate 00 into D (press CTL-1 to abort)

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.

```
>B
WELCOME TO THE 1802
RCA BASIC3 V1.1
(C)1981 RCA
C/W?
_
```

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).

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                          BASIC3 is now running.
:10 PRINT 355/113              Enter a BASIC program (hint: the / sign is CTL-9).
:20 END                        Note that BS (the Backspace key) now works. :-)
:RUN                           Run your program.
3.1416                         Here's the result (it's approximately Pi).
READY                          Ready for more...
:BYE                           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/BASIC3v11user.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$(n)
Machine language: sets R3=addr, P=3,			Relational:	FVAL($)
CALL(addr,n1,n2) X=2, R8=n1, RA=n2; return with SEP R5			= > < <> >= <=	LEN($)
			Hex: #FF = 255	MID($,start,n)
v=USR(addr,n1,n2) same, but sets v=R8			@FFFF = 65535	TAB(n)

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                       Read the current baud rate.
E7CE 0D                        0D = 9600 baud (the default).
>WE7CE 19                      19 = 4800 baud (change to 4800 baud)
>WE7CE 35                      35 = 2400 baud (change to 2400 baud)
>WE7CE 68                      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 work to abort the SAVE command.

```
>S8000 400                    Save 400 bytes of memory starting at 8000.
Ready to SAVE Program         This message is BRIEFLY displayed...
                              Then the screen goes blank while sending the data.
File Saved Successfully       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 command. Note: <ESC> on the VIP2K will not abort during the LOAD command; but you can type <ESC> on your PC to abort the loading.

```
>L                            Load an Intel hex file...
Ready to LOAD Program         This message is BRIEFLY displayed...
                              Then the screen goes blank while receiving data.
                              Tell your PC to start sending data.
File Loaded Successfully      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.

C run CHIP-8 C <CR>

CHIP-8 is a simple, easy-to-learn interpreter, like BASIC. But 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. See the Wikipedia page at https://en.wikipedia.org/wiki/CHIP-8 for CHIP-8 documentation and information. Marcel van Tongeren wrote this version for the VIP2K.

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 with the I command **before** you run CHIP-8 itself. If you enter the C command with no CHIP-8 program loaded, the message No CHIP-8 File Loaded is displayed and you are returned to the Monitor's >_ prompt.

I load 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                          Load a CHIP-8 Intel hex file...
Ready to LOAD Program       This message is BRIEFLY displayed...
                            Then the screen goes blank while waiting for data.
                            Tell your PC to start sending data.

File Loaded Successfully    This message is displayed when it's been received.
>MFEFD 1                    Let's check the CHIP-8 "Loaded" flag at FEFD.
>FEFD 80                    It will be 80 if successful (or 00 if unsuccessful).
>C                          NOW you can use the C command to run CHIP-8.
```

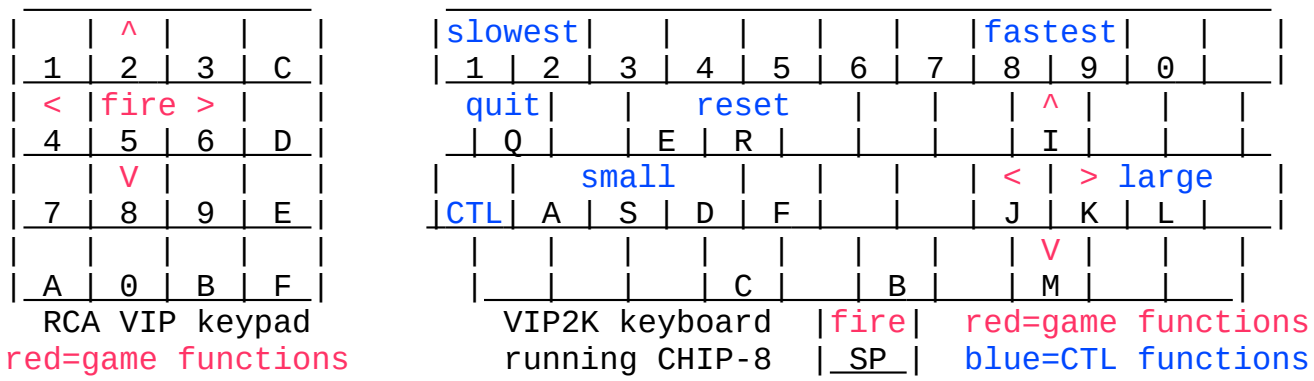
CHIP-8 Operation

Keyboard commands while CHIP-8 is running:

CTL-Q	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 (3x5 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.



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

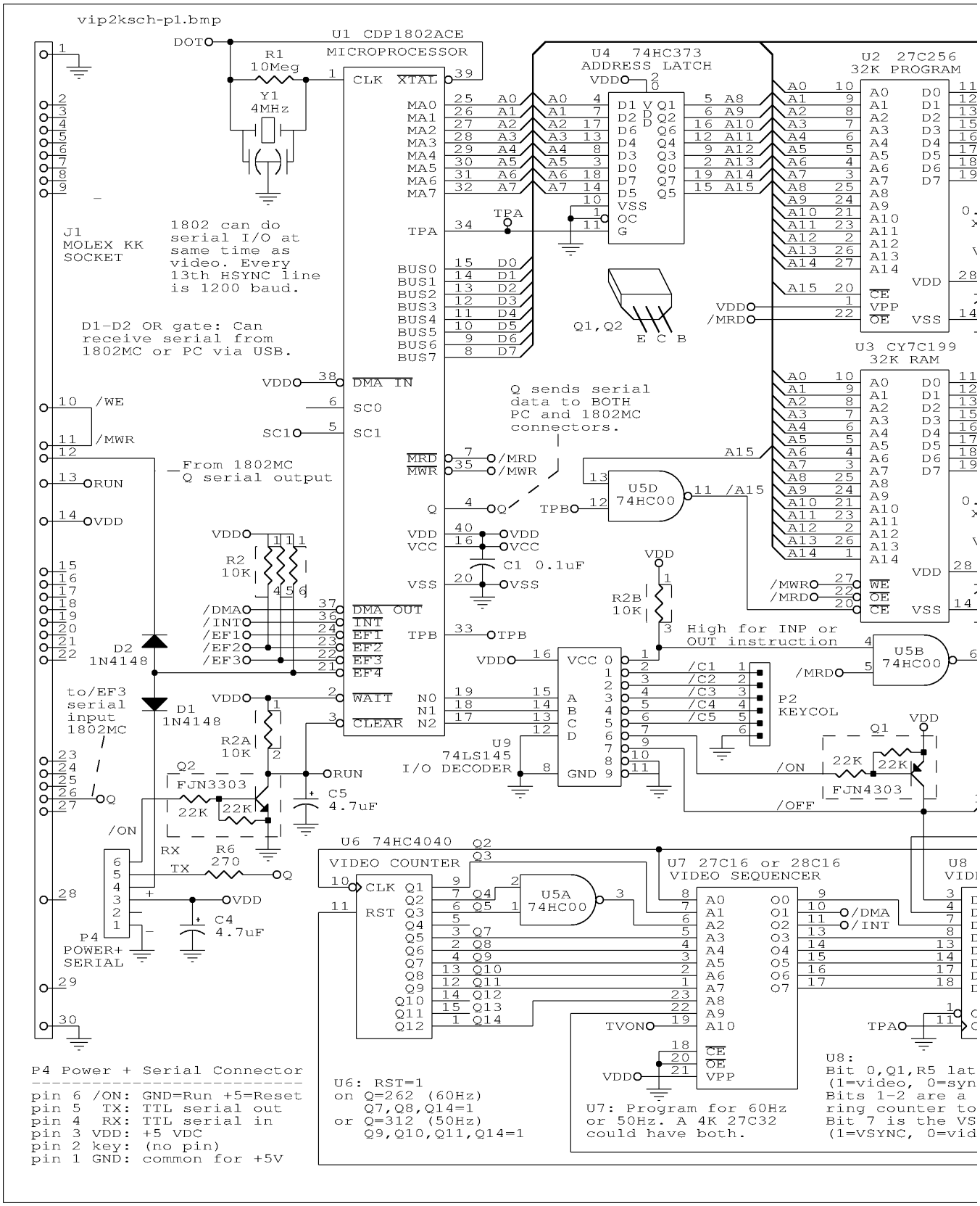
CHIP-8 Memory Map

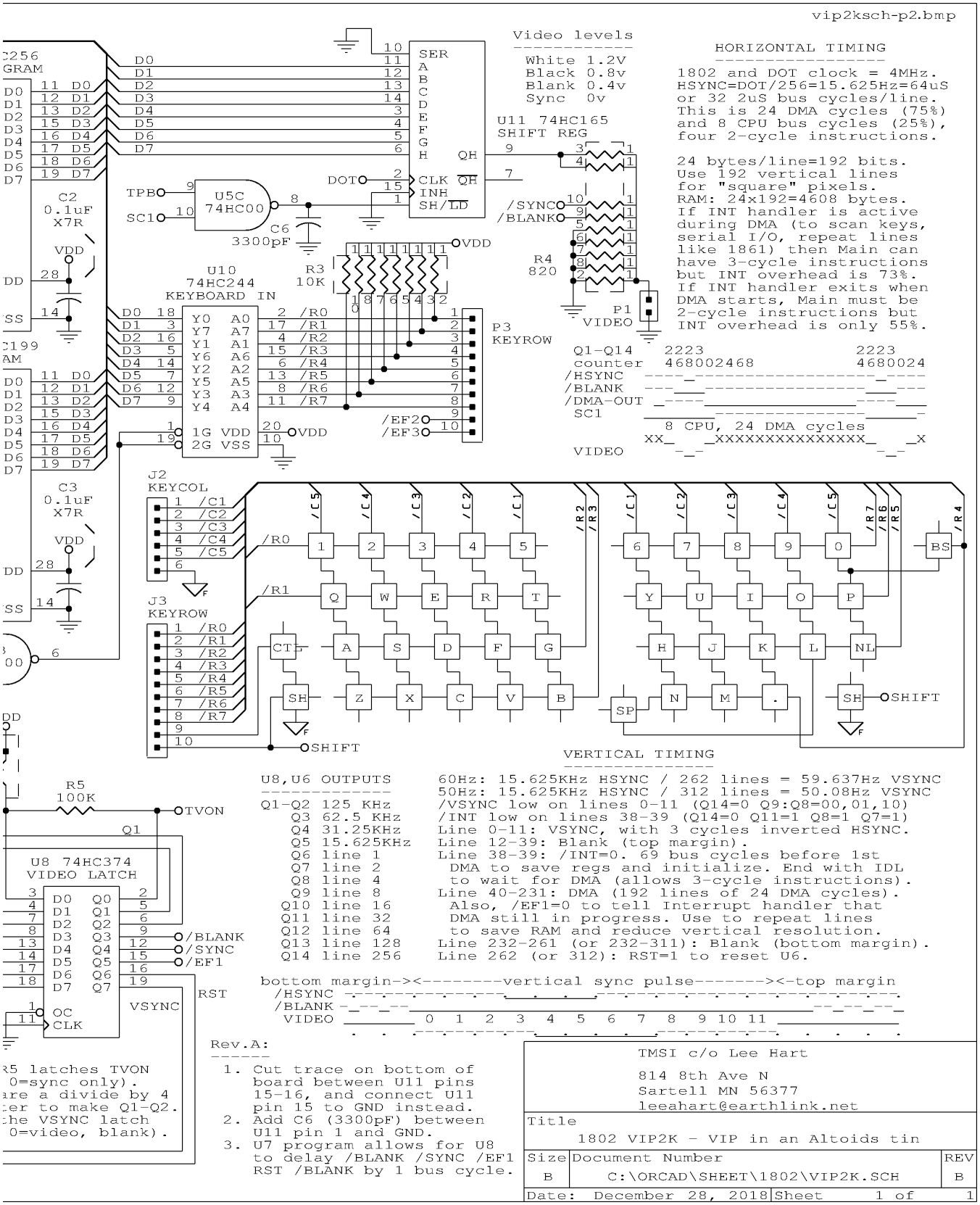
8000-81FF	Normally not used, but available for CHIP-8 code
8200-8FFF	CHIP-8 user space
E800-EC77	Video RAM in LARGE 3x5 pixel format:
E800-E84D	- Top 3 lines; not used by CHIP-8 (should always be 0)
E84E-EB8D	- CHIP-8 video screen
EB8E-EBF5	- Bottom 4 lines; used for some games like Pong
EBF6-EC77	- 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)
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

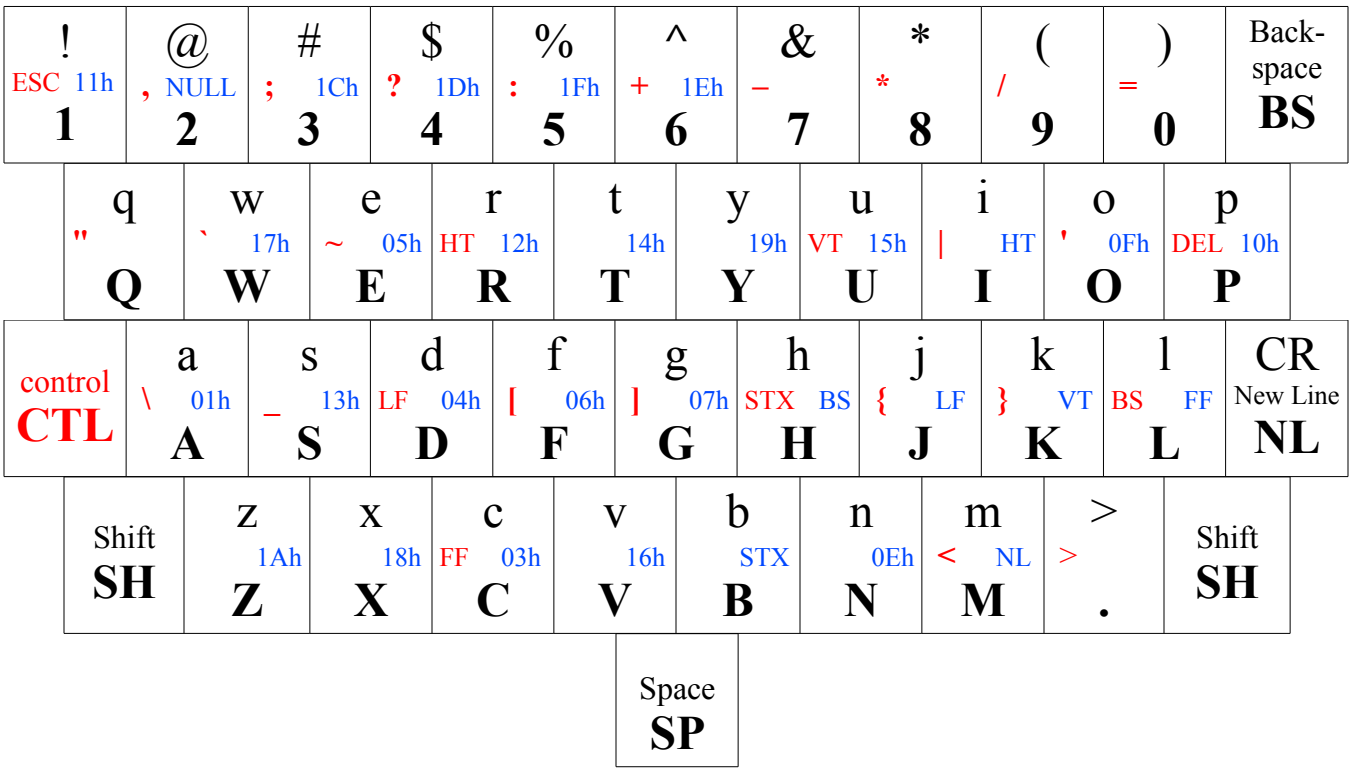
	Original VIP CHIP-8	**VIP2K CHIP-8**
User Space	0200-0EFF	8000-8FFF (starting address is 8200)
00aa, SYS 0aa		Call 1802 system routine at 70aa
0aaa, SYS aaa	Call 1802 system routine at aaa	Call 1802 system routine at 8aaa (aaa>0FF)
1aaa, JP aaa	Jump to address aaa	Jump to address 8aaa
2aaa, CALL aaa	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

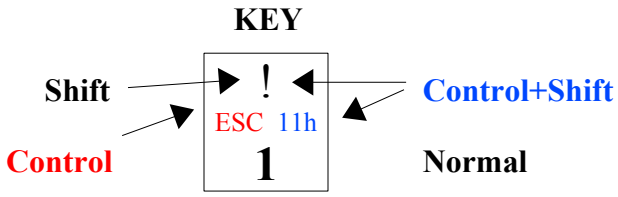



vip2ksch-p2.bmp
Video levels
White 1.2V
Black 0.8v
Blank 0.4v
Sync 0v
HORIZONTAL TIMING
1802 and DOT clock = 4MHz.
HSYNC=DOT/256=15.625Hz=64uS
or 32 2uS bus cycles/line.
This is 24 DMA cycles (75%)
and 8 CPU bus cycles (25%),
four 2-cycle instructions.
24 bytes/line=192 bits.
Use 192 vertical lines
for "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%.
U11 74HC165
SHIFT REG
U5C
74HC00
U10
74HC244
KEYBOARD IN
P3
KEYROW
P1
VIDEO
R3
10K
R4
820
C2
0.1uF
X7R
C3
0.1uF
X7R
C6
3300pF
J2
KEYCOL
J3
KEYROW
Q1-Q14 2223 2223
counter 468002468 4680024
/HSYNC
/BLANK
/DMA-OUT
SC1
8 CPU, 24 DMA cycles
VIDEO
VERTICAL TIMING
U8,U6 OUTPUTS
Q1 125 KHz
Q3 62.5 KHz
Q4 31.25KHz
Q5 15.625KHz
Q6 line 1
Q7 line 2
Q8 line 4
Q9 line 8
Q10 line 16
Q11 line 32
Q12 line 64
Q13 line 128
Q14 line 256
60Hz: 15.625KHz HSYNC / 262 lines = 59.637Hz VSYNC
50Hz: 15.625KHz HSYNC / 312 lines = 50.08Hz VSYNC
/VSYNC low on lines 0-11 (Q14=0 Q9:Q8=00,01,10)
/INT low on lines 38-39 (Q14=0 Q11=1 Q8=1 Q7=1)
Line 0-11: VSYNC, with 3 cycles 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 with IDL
to wait for DMA (allows 3-cycle instructions).
Line 40-231: DMA (192 lines of 24 DMA cycles).
Also, /EF1=0 to tell Interrupt handler that
DMA 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
/HSYNC
/BLANK
VIDEO
U8 74HC374
VIDEO LATCH
R5
100K
TVON
/BLANK
/SYNC
/EF1
RST
VSYNC
SHIFT
R5 latches TVON
0=sync only).
are a divide by 4
er to make Q1-Q2.
the VSYNC latch
0=video, blank).
Rev.A:
1. Cut trace on bottom of board between U11 pins 15-16, and connect U11 pin 15 to GND instead.
2. Add C6 (3300pF) between U11 pin 1 and GND.
3. U7 program allows for U8 to delay /BLANK /SYNC /EF1 RST /BLANK by 1 bus cycle.
TMSI c/o Lee Hart
814 8th Ave N
Sartell MN 56377
leeahart@earthlink.net
Title
1802 VIP2K - VIP in an Altoids tin
Size B
Document Number C:\ORCAD\SHEET\1802\VIP2K.SCH
REV B
Date: December 28, 2018
Sheet 1 of 1

Appendix B -- Keyboard layout and Key Codes



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



Blank positions return a value of 00h (NULL).
The SH (shift) and CTR (control) keys return no code on their own.