**OPERATING NOTES FOR VTL-2, vers. ‘Cu’ FOR UNEXPANDED 8k PAL-1 AND microKIM 6502 COMPUTERS**

VTL-2Cu is based on Jim McClanahan’s adaptation of Michael T. Berry’s 6502 port of Frank McCoy and Gary Shannon’s 768-byte interpreted high-level language for the Altair 680 in 1976.

All I (Dave Hassler) did to Jim’s ROM version is change the load/origin address to sit it in the top of an “unexpanded” PAL-1’s RAM and move the bottom of user RAM down a page, then recompiled. This version gives the programmer 3,247 bytes to work with, almost a full kilobyte more than Tiny BASIC running in the same environment (and almost as much user RAM as my VIC-20 had, back in the day!).

The reason for doing this is so owners of PAL-1s without expansion RAM and/or ROM cards can have another option when it comes to high-level programming. And if you’ve worked with Tiny, VTL-2 will be a snap: the concepts are almost identical (it took me a little more than an hour to get the hang of it), and the code comes out to roughly the same size as Tiny.

For a wealth of info on this version of VTL-2, see Jim’s github pages and check out Micheal’s fully-commented source code, with Jim’s notes. It’s an education! Also, for complete instructions on using VTL-2, I have included Frank’s updated documentation as a PDF.

**CHARACTERISTICS**

Load it from an ASCII upload of a MOS papertape, or from cassette (or whatever) via the Hypertape WAV file provided (load time is about 25 seconds for the 1,103-byte program). Enter the program at $0FB0 for a cold start, $0FC1 for a warm start.

After a user program upload (ASCII transfer via serial terminal), type a few <RETURN>s to leave the “enter program lines” mode and back to a direct mode “OK” prompt.

There is *absolutely no error checking* in VTL-2. When entering a line of code, backspace (or delete – CHR$(8)) will back up and erase a character from the buffer, while the ESCAPE key is the “wipe out the whole line” key.

The decimal numbers allowed are 0-65535, unsigned, and overflowing equations will wrap-around the 16-bit limit.

Finally, a 10 ms character delay and a 200ms line delay on serial transfers seems stable, keep your CAPS LOCK on, CTRL-C breaks, don’t enter <RETURN> alone ever on input, and best: it runs pretty darn fast! Have fun!

**THOUGHTS**

It would not be difficult to create a little load-n-save-to-tape program and call it from within VTL-2. Just save everything from $0000 to whatever is in the variable & -- inefficient, totally quick-n-dirty -- but it would work. Or maybe have your routine start at $0300, then store the last address of the load to $CC and $CD (the & location) and then warm start.

Areas of unused RAM: bottom of stack at $0100 (like usual), and only the bottom 72 bytes of Page 2 are (read: most often should be) used for the input buffer. These would be good places to tuck away ML routines. Also, this version of VTL-2 uses only $17E5 and $17E6 in the RIOT's RAM for temporary storage of registers; the rest of it is available for use.

**SYSTEM VARIABLES**

! -- Return address/line after last #= statement

" -- Starts ‘literal’ print; also addr of > ML jump

# -- Current line number (essentially, GOTO)

$ -- Single character input/output

% -- Remainder after last division

& -- Pointer to last byte of user program

' -- Random number between 0-65535

( -- Start of nested expression

) -- End of ... lots of stuff

\* -- Pointer to top of user RAM

< -- Points to a location in memory

@ -- Loads/stores a value from/into location <

> -- Holds a # to pass to ML routine and JSRs

? -- Decimal number input/output

: -- Defines which element of The Array used

; -- Suppresses CRLF in a literal print statement

# TINY BASIC\* to VTL-2c

**STATEMENTS**

**CLEAR -- &=768** *on a system that starts user RAM at $0300.*

**\*DIM (ARRAY) -- :X)** *where X is the Xth 16-bit word in the single, unnamed, 1-D array. e.g.,* 20 A=:6)  *or* 20 :B)=A\*C. *X may not equal 0. The equal sign can be on either side of the array statement. The size of the array is only limited to free user RAM at the top of a program.*

**GOTO/GOSUB -- #=** *the current line number. Effectively, GOTO and GOSUB. e.g.,* #=100 *is* GOTO 100*.*

**END -- )** *not required.*

**IF/THEN -- a mix of expressions, assignment, and #=** *E.g.,* 20 #=(A=B)\*100 *is the same as* 20 IF A=B GOTO 100 *in Tiny BASIC. If A=B, result is 1; if not, 0. If true, the current line number (*#=*) will become the value of the expression \* 100, creating a branch. Similar to BEQ and BNE in 6502 assembly. The line* 10 A=B=C *is a comparison: if B=C then A=1; if not, A=0.*

**INPUT -- ? and $** *when placed on the right side of an equals sign, allows input of a number (*? *- decimal, 16-bit) or a letter (*$*). E.g.,* 20 A=?  *user types a 7 and <RET> A will be 7. Or,* 20 A=$ *user enters F and <RET> and A will be 70 (the ASCII value of F).*

**LET --** *not required, as in BASIC.*

**LIST -- 0** *the number 0 by itself will list a program. Use* CTRL-A *to halt the output, which does not pause, just stops printing. Tap* <RETURN> *to finish the listing.*

**POKE/PEEK -- < and @ *<*** *points at a location in memory. When* @ *is placed on the left side of an equals sign, the number to the right is ‘poked’ into the memory location. When @ is on the right of = (e.g.* P=@*), < is ‘peeked’ and* P *takes that value.*

**PRINT -- ? and $** *when placed on the left side of an equals sign,* ?=X *prints a number (decimal, 16-bit) or a string enclosed with double quotes (*?="HI!")*, while* $ *prints a single letter of the ASCII value given:* $=X*. No automatic CRLF after a number is printed.*

**REM -- )** *any text typed after a ) is ignored by the interpreter.*

**RETURN -- #=!** *when a* #= *branches, its line number is placed in* !*. If no other* #= *is executed in the “subroutine,”* #=! *will return the program to the next line after the* #=*.*

**RUN -- #=1** *jump to the first line (or the first one found after line 1) and begin execution. Line 0 is not allowed.*

**\*(FOR/NEXT) -- not in Tiny BASIC, but constructed for VTL-2 exactly as in Tiny:** 10 I=1 20 ?=I 30 ?=”” 40 I=I+1 50 #=(I<11)\*20  *is this in KIM-1 BASIC:*  10 FOR I=1TO10:PRINTI:NEXT *Line 30 in the VTL-2 routine is “print a CRLF.”*

**FUNCTIONS**

**\*FRE(0) -- ?=\*-&** *prints the number of bytes between the top of user RAM (*\**) and the end of user program (*&*).*

**RND -- ='** *The ‘tick mark’ returns a random value between 0-65535. It’s suggested to “seed” the RND generator with a dummy equation (e.g.,* Z=Z+19*) before using the first time.*

**USR -- " and >** *set the ML address to “JSR” to in* " *, then branch there with the variable in > passed to the accumulator. E.g.,* 10 "=6144 20 >=4736 *if one wanted the routine called to have $12 in the ACC and $80 in the X register before it executes (#4736=$1280).*

**EXPRESSIONS**

*Some of these are “system variables” as well as operators. VTL-2 relies on context to know what’s what, as in the* INPUT/PRINT *pairs. The bottom five expressions are only available in version C of VTL-2. Without parentheses, math is performed strictly left-to-right.*

**+ - \* / =**

**<** *this is less than.*

**>** *this is equal-to-or-greater-than.*

**MOD %** *a single-digit division remainder*

**OR |** *E.g.,* A=$ | 128 *means “get a character and set the high bit.”*

**XOR ^ AND &**

**(THEN) [** *A[B returns 0 if A=0, otherwise returns B.* **(ELSE) ]** *A]B returns B if A=0, otherwise 0.*