Foenix Rising
A newsletter for developers, users, and enthusiasts of Foenix Retro Systems products

(FLASH !)

This article contains no shortage of nostalgia. We discuss the state of word processing in 1984, one man and a publication's efforts to change the game, and ultimately, the reason why we are dusting off 40 year old code for integration into a Foenix Retro product. Yes, we're talking about COMPUTE! Publication's *SpeedScript*, written by Charles Brannon, an essential piece of their magazine and book publishing history.

We will conduct a deep dive into the inner workings and features of SpeedScript including discussing the work required to port it to the F256 to suit our selfish needs.

Sept. 2023 / F10

Re-use, recycle, reduce: dusting off and improving an early example of free software

Word Processing circa 1984

In October of 1983, Microsoft released Microsoft Word version 1.0 for DOS. It was an early example of WYSIWYG and despite Windows not yet being invented, it supported a mouse; Of course in this early day, what you saw was ≠ to what you got on output; it only displayed bold, italics, and underline text; fonts were not yet supported on-screen and proportional spacing had a ways to go.

In November of 1983, PC Magazine distributed a DOS demo version of Microsoft Word on diskette and it was included with the periodical making PC Magazine the first publication to do this sort of thing large scale.

Meanwhile, in the land of 8-bit microcomputers, scores of word processors were evolving and students and professionals world-wide were seeing real productivity boosts. This pace would continue for years to come.

The Word Processor was an early example of a killer app for home use; a panacea for households wealthy enough to afford a computer and an inexpensive dot-matrix printer loaded with tractor feed paper.

EasyScript for the Commodore 64 was my first foray in this area (and probably the only reason any of my schoolwork made it across the finish line); numerous products borne from CP/M platforms right up to the big daddy of native typesetting tools on Unix Systems were being iterated upon at a rapid rate.

It was by this time that COMPUTE! magazine, already dominating the home computer serial publication market, endeavored to create a lightweight, zero-cost application distributed as machine code along with a near-foolproof data-entry system (MLX) for 8-bit platforms.

There must have been a debate over whether the release of a viable software 'product' in this particular vertical would be balked at by advertisers that paid Compute Publishing's bills; but by the time SpeedScript was released, ABC Publishing owned the publication and subscribership was approaching 500,000! By the time SpeedScript 3.2 was released, numerous titles such as PaperClip, Word Pro, HesWriter and others were reaching the peak of their 'paying' distribution potential and software piracy was rampant. SpeedScript was feature packed and grew a large fanbase; so large that retro enthusiasts today still speak fondly of it.

In this issue, we will dive into Charles Brannon's SpeedScript platform including an internals discussion and our plans to repurpose some of the code for our own on-platform, native editor for the Foenix F256.

If you missed the last issue, here is a recap of what we've done so far and where we are heading:

last issue
- ✓ Write a memory examination utility from scratch, but tailored for the F256 platform. We will do so in support of a fork of a kernel environment that is also being developed; stay up to date on my YouTube channel: 8-Bit Wall of Doom) for more insight.
- ✓ Take Steve Wozniak's original Apple I disassembler as published in Dr. Dobb's journal, and port it to the F256.

this issue
- ✓ Port and integrate Charles Brannon's SpeedScript editor (because it is tiny, refined, and well documented) then discuss planned [and completed] enhancements.

next issue / later
- Modernize our disassembler to recognize the full WDC 65C02 instruction set (8 new instructions plus added addressing modes for the original MOS6502 opcodes).
- Port and integrate a File Manager with 'launch' capabilities from my own 6502 Source code (1986)
- Consider if a simple version can run under TinyCore/ MicroKernel and discuss what a minimum viable kernel ought to include from a services perspective.

It's worth noting that the functionality detailed above has yet to address the merging of this code with an enhanced version of the OpenFNXkernal (discussed on the 8-Bit Wall of Doom YouTube channel), but that will come later.

For now, we will maintain focus on delivering the MVP of our editor (a fork, if you will, of SpeedScript), finish the disassembler code, and subsequently, tie-up load and save features (which are numerous and will be discussed at length in future; the detail is important).

Retro Flashback: "A **COMPUTE! Books** Publication**"**

These words adorned the cover of dozens of books in the 1980s; treasures for young and older computer hobbyists across 8-bit platforms and beyond. The average publication weighed in between 250 and 350 pages and included scores of type-in programs, tables, teachings, and inspiration. Many of us had a collection. They were the '80s equivalent of the O'Reilly Press books that IT professionals stockpiled in the '90s.

In the early years, books of this nature tended to be highly edited compilations of magazine articles (see Assembly Lines) but soon, publishers were investing in new and highly focused content of all kinds. The bad news is that all are out of print; the good news is that just about all of them are available on archive.org and there is plenty of stock on the resale market at reasonable cost.

Two of my favorites were *Mapping the Commodore 64* (1984; Sheldon Leeman) and *Machine Language for Beginners* (1983; Richard Mansfield).

'Mapping' is a well known and an indispensable resource for 8-bit ATARI, Commodore (VIC, 64, 128), and even the IBM PC. And while I haven't seen all of them, those that I've had access to convey not only the 'what', but the 'why' (important) and the 'how' (to use).

But to me, Mansfield's ML for Beginners book is the most special because I thank its preface for inspiring me to immerse myself in assembly language at a young age.

The book begins with a simple premise: "**Something amazing lies beneath BASIC**". And across the next two pages, the author's own experience learning 6502 Assembly Language is discussed including detail of his investing ~80 hours (3-4x longer than his estimate) writing his first large scale, all-assembly language program. The passage triumphantly concludes:

"**I am probably more proud of *that* program than any other I've written.**"

Of course, Mansfield was not talking about a Word Processor, he was referring to a space-invader style shooter, complete with rows of aliens, sound, and more.

After working at Compute Publications, Mansfield went on to write dozens of books, many related to Visual Basic, .NET development, and other Microsoft products; have a look on Amazon for a partial view of his work.

I trust that he is well. He should be ~77 years old by now and based on my research, was last interviewed in 2016 when he shared a few tidbits about his life including the fact that he was an English major. Upon leaving College, he felt he only had two options for employment: "teach or write" and he couldn't see himself teaching. I'm sure I speak for many when I say that the industry was better thanks to his contribution.

My first machine language code was more modest (6 bytes). It was entered 15 minutes after I returned home from buying his book and it blew my mind:

```
>C000  EE 01 D0 4C 00 C0
```
(anyone?)

Enter (and *entering*) **SpeedScript**

In 1985 Compute Publications released a Charles Brannon book, compiled to support their highly successful SpeedScript Word Processor campaign.

The book focused on version 3.1 for the original platforms, the Commodore 64 and VIC-20, and it was short compared to other titles (about 165 pages).

The first 30 pages are quite useful. Chapter 1 introduces the basics and discusses features, describing the theory of use. Recall that in 1984 and 1985, Word Processing was a relatively new frontier for many users and the conventions that we've now grown accustomed to such as ctrl-right for next word, END key for end-of-line, etc. were not yet established. The text conveyed the mechanics of "how to use" in addition to why a given feature was significant. The bad news was, on-screen help was non-existent and there were no menus.

To add to the challenge, each application in the early days had a different command scheme which diverged further between ports of the same software across platforms. Some programs (SpeedScript was one) leveraged CTRL-shifted keys and function keys; others used a combination of meta-keys including the Commodore key. In short, the first section of the SpeeddScript book is a *required* read if you plan on using the platform as delivered. It's also a required read if you have any hope un understanding the code, should you wish to enhance it, as we will.

Chapter 2, on the other hand, is about a useful as a chocolate teapot. It is 60 pages in length and is 100% focused on Compute's [then] new data entry utility, MLX, including data. For those unaware, MLX was an improvement over the 'old' way of entering machine language: poked bytes from data statements. It included a checksumming system which made data entry nearly foolproof (that is, after you've keyed in and corrected about 100 lines of BASIC code, the MLX utility, itself!).

For our purposes, the interesting portion of the book begins on page 95 and is titled "Commodore 64 Source Code". After two pages of intro material, 30 pages of impossibly small assembly code follows. And it is laid out two columns per 1/2 sheet sized paper with occasional comments. (see the bottom of pg. 6)

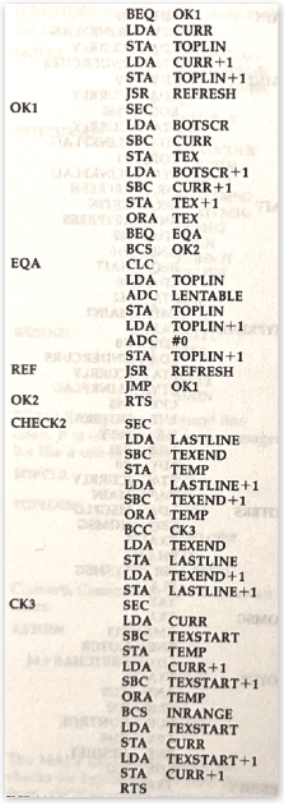
I typed it in; slowly and painfully over the course of three evenings until my eyes were bleeding. In total, I keyed in ~2,000 lines of source before familiarity led me to realize I could omit sections of code that I had no use for. The experience was a bit like reading a book for the first time, knowing how it would end. This realization probably saved me about 1,000 lines of typing!

The book concludes with 4 pages of an add-on/mod called "ScriptSave" which saves the document in memory every 10 minutes. This is followed by a cut-out function key template, a cut-out Quick Reference card, and a cut-out cheat sheet containing the print formatting

commands. Then there is a single page index and 8 blank pages headed with "Notes".

Coding style, features, and 'undoing' Commodore

Brannon's code reminds me of my own code from 1984; meaning, it's amateurish. Labels are oddly named and there is a lack of comments except by major section.



```
        BEQ   OK1
        LDA   CURR
        STA   TOPLIN
        LDA   CURR+1
        STA   TOPLIN+1
        JSR   REFRESH
OK1     SEC
        LDA   BOTSCR
        SBC   CURR
        STA   TEX
        LDA   BOTSCR+1
        SBC   CURR+1
        STA   TEX+1
        ORA   TEX
        BEQ   EQA
        BCS   OK2
EQA     CLC
        LDA   TOPLIN
        ADC   LENTABLE
        STA   TOPLIN
        LDA   TOPLIN+1
        ADC   #0
        STA   TOPLIN+1
REF     JSR   REFRESH
        JMP   OK1
OK2     RTS
CHECK2  SEC
        LDA   LASTLINE
        SBC   TEXEND
        STA   TEMP
        LDA   LASTLINE+1
        SBC   TEXEND+1
        ORA   TEMP
        BCC   CK3
        LDA   TEXEND
        STA   LASTLINE
        LDA   TEXEND+1
        STA   LASTLINE+1
CK3     SEC
        LDA   CURR
        SBC   TEXSTART
        STA   TEMP
        LDA   CURR+1
        SBC   TEXSTART+1
        ORA   TEMP
        BCS   INRANGE
        LDA   TEXSTART
        STA   CURR
        LDA   TEXSTART+1
        STA   CURR+1
        RTS
```

It's not much different than the BASIC code that found its way into magazines and manuals in the early '80s. You'd think that editors would have renumbered lines or cleaned up code for clarity before publishing and of course, you would be mistaken.

My absolute favorite classic example of this was cited in issue #1 of this Newsletter, reprinted on pg. 7 below. See the footnote where I exclaim: "Come on Commodore!".

From a functionality and design perspective, Brannon incorporated some unusual features coupled with a few absolutely brilliant ideas. An example of the former:

Pressing shifted RUN/STOP inserts 255 spaces by calling the `LOTTASPACES` subroutine. Of course*.

Meanwhile, CTRL-X transposes the current character with the subsequent character. Why would you want or need such a feature? I suppose if you routinely typed received as recieved; assuming you bothered to move the cursor left two spaces this would be handy?

Then there's CTRL-A which flips alpha case, not of the current word or highlighted passage (the concept of highlighting did not exist), it only worked on the character under the cursor. Again, quicker to type a backspace and the character you really wanted.

On the other hand, SpeedScript was indeed built for speed (footnote below, aside). An optimized screen update routine (`REFRESH`) does its thing every cursor flash whether data has changed or not. How does it get away with updating up to 1,000 characters twice a second? By avoiding kernal routines such as `CHROUT` and instead, rendering text directly from stored screen codes (at least in the Commodore versions).

Screen codes are values that are `POKE`'d into screen memory. By avoiding all of the PETSCII conditionals that would otherwise test for reverse field, cursor controls, color attribute change, etc. tons of clock cycles are saved. Anything multiplied by 1,000 turns into a big number!

Another somewhat progressive design decision was to save document text in this native screen code format and do so directly with a kernal routine (`SAVE`) as if the data was an executable program. By doing this, Brannon greatly reduced the amount of code required and provided the benefit of allowing a direct load, again, with zero added code. The scheme only required a `SETLFS`, `SETNAM`, and a `LOAD` strapped with the appropriate memory address which was either the start of text memory or at a chosen insert point. Smart.

There were also a number of oddities thrown in. Brannon felt it necessary to leverage a raster interrupt on the 9th visible scan line to create a colored status and command bar that was a different color than the rest of the screen. He also injected some unusual error trapping code which dual purposed a saved character color to identify state.

The port to the Foenix F256 platform was relatively straight forward once all of the Commodore 64 memory dependent code and subroutines were removed. This also included routines which were not desirable for our use-case such as printer driver code, the print formatting scheme (headers, footers, and margins), NMI code, a custom IRQ shim, the RUN/STOP trap, Jiffy clock tracking, the list goes on.

This work reduced the memory footprint from ~6,158 bytes to less than 4K (4,048 byes) for the bare bones version. But don't worry, we will build it back up by introducing improved editing features, on-screen help, and get us set up for functionality that dovetails nicely into our intended target platform; nanoCLI, which is rooted in a Commodore-like kernel.

Ultimately, our goal is to support Foenix specific features such as the 80 x 60 text mode, to potentially build in a rudimentary editor-driven assembler function, to possibly support copy/paste with split screen editing, but also to fit within an 8K FLASH bank such that *residency* is possible, recalled by a single keystroke.

Planned Foenix F256 specific changes

Since we borrow the codebase from SpeedScript, our version (nanoEDIT) will have a similar look and feel. It will leverage a top of screen reverse field status bar for displaying information such as 'free memory', for entry of search terms, or program load/save parameters. But we will add a lower status bar (line 30 when in double-high mode or line 60 otherwise). The lower line displays the line and column number, whether in 'insert' or 'overwrite' mode, the percentage through the file, and other information.

We've also completely re-worked the command key structure so features bound to function keys are common to other nanoCLI features. The revamped CTRL key map also has added functions and a handful of reassigned keys that are closer in behavior to modern

---

* there is actually a very good reason why you might need this function on a 1 MHz. system. See page 6 (epilogue).

text editors. The table on the following page summarize the growing list of functions with just a pair of commands on the drawing board (in grey), not part of our minimal viable version.

Note the '*' which indicates an original SpeedScript feature that we may or may not have bound to a different key.

CTRL plus:

@ = help or *special* command
^ = move start-of-line (or 'a')
$ = move end-of-line (or 'e')
b = border & screen color increment*
c = copy to kill buffer (W, S, B, E)
d = delete (W, S, B, E) to kill buffer*
h = hunt*
i = tab for movement (or TAB key to insert)
l = character color*
o = overwrite / insert toggle*
p = paste kill buffer
r = redo
s = search with replace
u = undo
= = mem remaining / consumed, and stats*

HOME followed by:

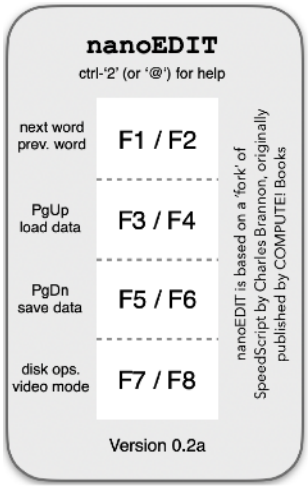
HOME = start of doc (jump to)
END = end of doc (jump to)
LEFT = start of screen line (same as '^')
RIGHT = end of screen line (same as '$')
UP = top of screen (upper left)
DOWN = bottom right of screen text
(any other key escapes)

shifted-HOME = clear memory (w/confirm)*
cursor keys, delete, DEL, INS keys as you would expect

Function Keys

Of course there's a function key template! No self respecting '80s application should be without one.

In our case, however, what you see to the right is just a prop. The F256K has standard sized keys for this purpose and those using PS/2 keyboards have the traditional row which extends F1 through F12.



Regardless, the depiction to the right is accurate as of the current version in build. With a fair amount of work still left to do, we will cover these feature details as additional functionality develops.

More SpeedScript internals: The Cursor conundrum

The Commodore kernal has a handful of vectors and functions that depend on zero page addresses and timers in order to enable and track the familiar blinking cursor.

It is important to consider that a cursor is a logical entity, pointed to by *x* and *y* counters representing the location where the *next* char will be written. Of course this requires math in order to compute screen memory addresses and that costs machine cycles.

Print control routines affect counters based on the type of character being printed such that backspace, cursor controls, clear screen, and printing off the edge of the screen are accounted for differently.

To illustrate this behavior with a simple example, consider the act of clearing the screen by printing a `CHR$(147)` and subsequently printing the letter 'A'.

You might print this from Commodore BASIC with the following command:

```
PRINT "{CLR}A"
```

… or a `chr$(12);"A"` from SuperBASIC.

In each case, screen memory is filled with spaces, and the cursor is moved to the top left corner of the screen where the 'A' is printed. And since the PRINT statement ends without the ';' character, a carriage return is asserted, moving the logical cursor down one additional row and to the first column. The resulting calculation is row (1) * SCREEN_WIDTH (40) + column (0) + SCREEN_BASE ($0400 on the Commodore 64 or $C000 on a Foenix F256K). Maths.

Contrary to terminals and some microcomputers, Commodore computers and Foenix systems do not display a physical cursor while printing. The visual or physical cursor 'character' on the other hand is controlled by a series of counters and status bytes to manage the rate at which the char in the current logical cursor position will be swapped back and forth from its reverse field counterpart.

On the Commodore 64, this is accomplished using several memory locations including:

$C9 and $CA (X and Y position)
$CC blink enable flag
$CD countdown to flash (jiffys starting with 20)
$CE character under the cursor
$CF a flag indicating if the last flash was 'on' or 'off'

With this background behind us, I need to share that SpeedScript uses none of this!

Instead, it manages its own cursor by watching jiffy clock timers at locations $A0 - $A2 and changing the character in memory at the insertion point to its %10000000 (hex $80) complement using the `EOR` opcode and after a time delay, doing it again.

SpeedScript leverages BLINKFLAG and a single byte memory location which temporarily holds the character 'under' the cursor in UNDERCURS.

Managed within the tight MAIN keyboard input loop, this routine also calls REFRESH to render document memory on-screen approximately twice a second. And this dance goes on continuously until you press a key.

* original SpeedScript features

If you scan the SpeedScript code, you will notice that there are numerous CHROUT calls, and even a number of jsr GETIN calls. But why?

The answer is, SpeedScript uses these routines; not for displaying document text, but for rendering messages on the upper status line and also to display disk directories called for by CTRL-4 or the DOS Wedge.

Interestingly, the act of loading, saving, or verifying a file produces no program generated output whatsoever; instead, SpeedScript relies on kernal messages printed directly after modifying the kernal message control verbosity setting located at $9d.

Foenix platforms, thanks to the VICKY FPGA, enjoy a hardware cursor that merely needs to be enabled and optionally may be modified to flash at a given rate, alternating an identified glyph or standard character with whatever character happens to be present in a given location on the screen. It is all done in hardware and without the use of reverse field characters.

But this leaves us in a bind. We don't want to rewrite too much (or any) of the core SpeedScript code, and we do not have a reverse field character set. Now what?

The answer is, we will make our own and then leave the SpeedScript code as is. Aside from the 26 bytes of code below, this approach requires zero additional memory. Unlike Commodore platforms, the Foenix font (there are two of them) occupy a read/write bank of memory which is instantiated at system boot and does not get in the way of, or take the place of the RAM beneath.

The code below copies exclusive OR'd versions of standard printable characters to their upper ($80 complement) position within the $C000 font memory bank. As an example, ASCII 32 (a space) will be rendered in ASCII 160 as a solid reverse field block.

For completeness, I should mention that VICKY's character video engine does allow differently colored foreground and background characters, so why not just use that to create reverse field? Because it will cost more cycles than this approach, and again, we aim to leave as much of the original code in tact as possible. It is my hope that an enterprising individual will find value in our version and back-port it to another system (perhaps the Commodore 64 :)

Here is the subroutine, executed during initialization during the first run of the program, only.

```
            ldx #$00
fontouter   inc fontinner+2
            inc fontupper+2
            ldy #$00
fontinner   lda $bf00,y
            eor #$ff      ; makes 0's 1 and 1's 0
fontupper   sta $c300,y
            iny
            bne fontinner
            inx
            cpx #$04
            bne fontouter
```

Note that we are stopping short of changing other glyphs or graphic characters built into the Foenix set. We are not adopting the PETSCII set entirely. But there is one exception.

The Commodore version of SpeedScript uses a left pointing arrow '←' (PETSCII 95 decimal). Frankly, I was never crazy about it. I've always preferred the special symbol used in the ATARI 8-bit port of SpeedScript, so I used the following code to redefine the character:

```
            ldx #$08
arrowloop   lda returnchar-1,x
            sta $c0f7,x
            dex
            bne arrowloop
```

The following data was modeled using Ernesto's `spredit-jr` in 8 x 8 mode:

```
returnchar  .byte %00000000
            .byte %00010000
            .byte %00001000
            .byte %00001000
            .byte %01010000
            .byte %01100000
            .byte %01110000
            .byte %00000000
```

The combination renders newline symbols as follows:

The only other tweak is something that I very much liked on ATARI platforms; the fact that the cursor does not normally flash. No added code or data for that one, I removed a few instructions and saved a few bytes.

Next time: More to talk about

We have two features planned but not yet coded and between now and the next issue, we will be working on these. One is trivial, the other will require some design and experimentation.

a. Status line and enhanced memory-available tracking; nanoEDIT will use a lower status line to display telemetry information and we will use a timer to update this data every 30 seconds or when text is modified. The other thing the timer routine will do is count minutes and then issue an autosave function (if enabled).

b. But the real work will be to implement an undo/redo journaling feature leveraging an out-of-view 8K block of RAM. This will not cost us any primary memory, we'll slyly bank it in then out as edits are made. At least that's the plan.

That's all for now. Stay tuned for additional development in this area in the October timeframe and don't forget to keep up with development of our kernal and SuperBASIC series on the 8-Bit Wall of Doom YouTube channel.

Epilogue - more retro; more SpeedScript

Those that remember the original magazine probably remember what COMPUTE! (then “COMPUTE.”) originally looked like. It was first published as 1979 turned to 1980 and was primarily focused on the Commodore PET.

Foenix Rising took a look at Issue #2 in July of last year and we’ve attached a reprint of our article at the end of this article (pg. 8-11). If you are interested in nostalgia, it’s a look at the beginning of a company that became a publishing powerhouse; starting as a one-person newsletter and acquired for $18 Million dollars by ABC Publishing just 3 1/2 years later.

And in case you were wondering what became of SpeedScript, you may be interested to learn (as I recently did) that it still has an active fanbase and ongoing development.

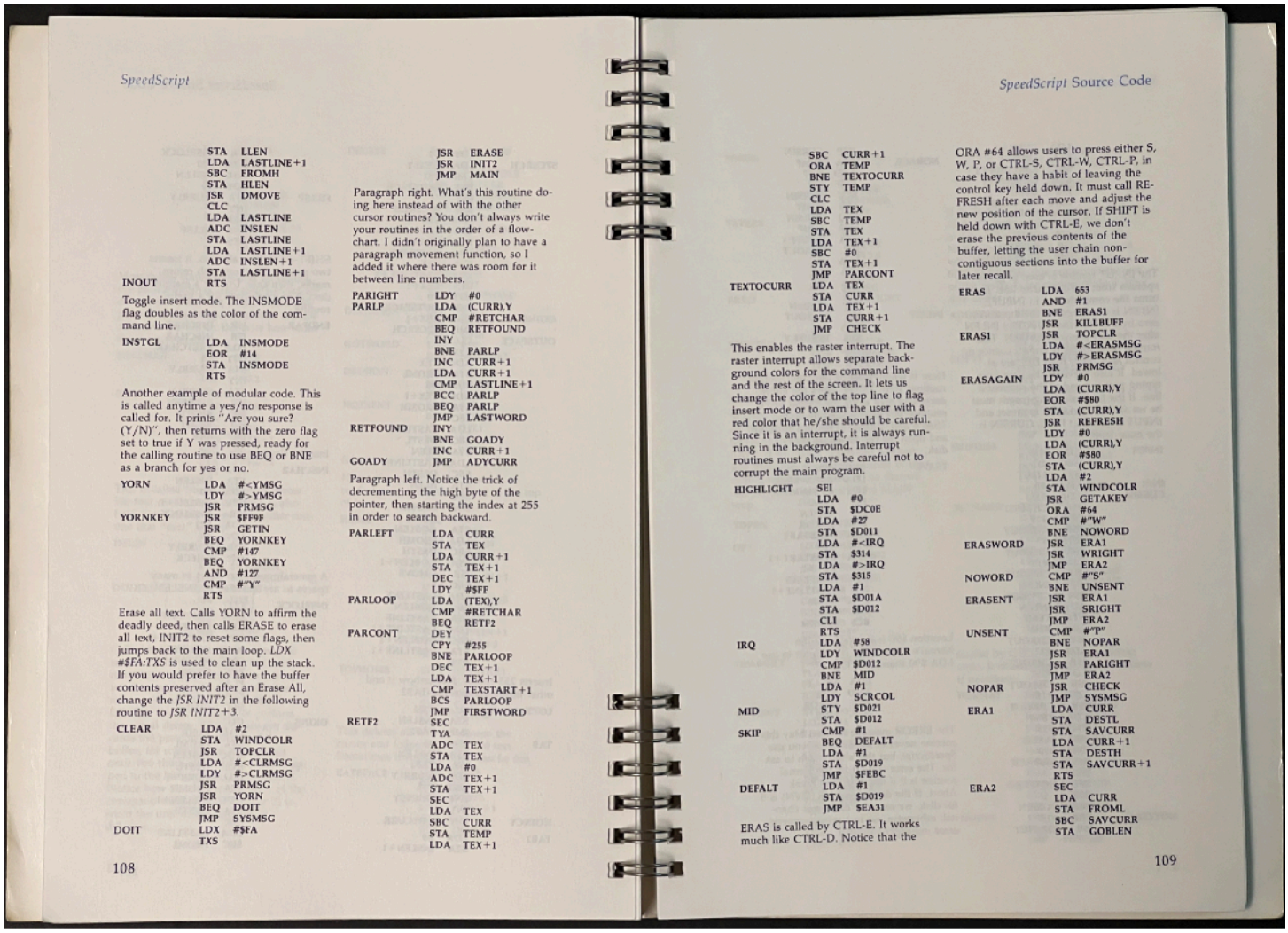
At VCF Midwest 18 (last weekend), I was introduced to a fellow enthusiast that did something comparable to my initial effort; he typed in the entire source from the archive.org copy, eventually purchasing Brannon’s book, as I did.

His aim is to maintain the source and functionality, to incorporate all of the available add-on patches published by COMPUTE! in the intervening years, and to do so in a modern Makefile so others can build any of the versions desired with byte-for-byte accuracy. I’ll publish his github once I get my hands on it.

And finally, I wanted to share the reason why the LOTTASPACES function exists: by Charles Brannon’s own admittance, working in insert mode can be slow when at the beginning of a large document. SpeedScript has to move all of memory forward character-by-character and this can get expensive on a 1 MHz. machine. The remedy? Open up a large space at once (“enough room for a few sentences”).

In our port, we benefit from a 6 MHz. CPU and faster memory, but also have access to VICKY’s DMA engine which we may or may not use. Stay tuned!

Just for fun - a closer look at the way code is represented in the SpeedScript book (2 pages of 30)



SpeedScript

```
            STA   LLEN
            LDA   LASTLINE+1
            SBC   FROMH
            STA   HLEN
            JSR   DMOVE
            CLC
            LDA   LASTLINE
            ADC   INSLEN
            STA   LASTLINE
            LDA   LASTLINE+1
            ADC   INSLEN+1
            STA   LASTLINE+1
INOUT       RTS
```

Toggle insert mode. The INSMODE flag doubles as the color of the command line.

```
INSTGL      LDA   INSMODE
            EOR   #14
            STA   INSMODE
            RTS
```

Another example of modular code. This is called anytime a yes/no response is called for. It prints "Are you sure? (Y/N)", then returns with the zero flag set to true if Y was pressed, ready for the calling routine to use BEQ or BNE as a branch for yes or no.

```
YORN        LDA   #<YMSG
            LDY   #>YMSG
            JSR   PRMSG
YORNKEY     JSR   $FF9F
            JSR   GETIN
            BEQ   YORNKEY
            CMP   #147
            BEQ   YORNKEY
            AND   #127
            CMP   #"Y"
            RTS
```

Erase all text. Calls YORN to affirm the deadly deed, then calls ERASE to erase all text, INIT2 to reset some flags, then jumps back to the main loop. *LDX #$FA:TXS* is used to clean up the stack. If you would prefer to have the buffer contents preserved after an Erase All, change the *JSR INIT2* in the following routine to *JSR INIT2+3*.

```
CLEAR       LDA   #2
            STA   WINDCOLR
            JSR   TOPCLR
            LDA   #<CLRMSG
            LDY   #>CLRMSG
            JSR   PRMSG
            JSR   YORN
            BEQ   DOIT
            JMP   SYSMSG
DOIT        LDX   #$FA
            TXS
            JSR   ERASE
            JSR   INIT2
            JMP   MAIN
```

Paragraph right. What’s this routine doing here instead of with the other cursor routines? You don’t always write your routines in the order of a flowchart. I didn’t originally plan to have a paragraph movement function, so I added it where there was room for it between line numbers.

```
PARIGHT     LDY   #0
PARLP       LDA   (CURR),Y
            CMP   #RETCHAR
            BEQ   RETFOUND
            INY
            BNE   PARLP
            INC   CURR+1
            LDA   CURR+1
            CMP   LASTLINE+1
            BCC   PARLP
            BEQ   PARLP
            JMP   LASTWORD
RETFOUND    INY
            BNE   GOADY
            INC   CURR+1
GOADY       JMP   ADYCURR
```

Paragraph left. Notice the trick of decrementing the high byte of the pointer, then starting the index at 255 in order to search backward.

```
PARLEFT     LDA   CURR
            STA   TEX
            LDA   CURR+1
            STA   TEX+1
            DEC   TEX+1
            LDY   #$FF
PARLOOP     LDA   (TEX),Y
            CMP   #RETCHAR
            BEQ   RETF2
PARCONT     DEY
            CPY   #255
            BNE   PARLOOP
            DEC   TEX+1
            LDA   TEX+1
            CMP   TEXSTART+1
            BCS   PARLOOP
            JMP   FIRSTWORD
RETF2       SEC
            TYA
            ADC   TEX
            STA   TEX
            LDA   #0
            ADC   TEX+1
            STA   TEX+1
            SEC
            LDA   TEX
            SBC   CURR
            STA   TEMP
            LDA   TEX+1
            SBC   CURR+1
            ORA   TEMP
            BNE   TEXTOCURR
            STY   TEMP
            CLC
            LDA   TEX
            SBC   TEMP
            STA   TEX
            LDA   TEX+1
            SBC   #0
            STA   TEX+1
            JMP   PARCONT
TEXTOCURR   LDA   TEX
            STA   CURR
            LDA   TEX+1
            STA   CURR+1
            JMP   CHECK
```

This enables the raster interrupt. The raster interrupt allows separate background colors for the command line and the rest of the screen. It lets us change the color of the top line to flag insert mode or to warn the user with a red color that he/she should be careful. Since it is an interrupt, it is always running in the background. Interrupt routines must always be careful not to corrupt the main program.

```
HIGHLIGHT   SEI
            LDA   #0
            STA   $DC0E
            LDA   #27
            STA   $D011
            LDA   #<IRQ
            STA   $314
            LDA   #>IRQ
            STA   $315
            LDA   #1
            STA   $D01A
            STA   $D012
            CLI
            RTS
IRQ         LDA   #58
            LDY   WINDCOLR
            CMP   $D012
            BNE   MID
            LDA   #1
            LDY   SCRCOL
MID         STY   $D021
            STA   $D012
SKIP        CMP   #1
            BEQ   DEFALT
            LDA   #1
            STA   $D019
            JMP   $FEBC
DEFALT      LDA   #1
            STA   $D019
            JMP   $EA31
```

ERAS is called by CTRL-E. It works much like CTRL-D. Notice that the ORA #64 allows users to press either S, W, P, or CTRL-S, CTRL-W, CTRL-P, in case they have a habit of leaving the control key held down. It must call REFRESH after each move and adjust the new position of the cursor. If SHIFT is held down with CTRL-E, we don’t erase the previous contents of the buffer, letting the user chain noncontiguous sections into the buffer for later recall.

```
ERAS        LDA   653
            AND   #1
            BNE   ERAS1
            JSR   KILLBUFF
ERAS1       JSR   TOPCLR
            LDA   #<ERASMSG
            LDY   #>ERASMSG
            JSR   PRMSG
ERASAGAIN   LDY   #0
            LDA   (CURR),Y
            EOR   #$80
            STA   (CURR),Y
            JSR   REFRESH
            LDY   #0
            LDA   (CURR),Y
            EOR   #$80
            STA   (CURR),Y
            LDA   #2
            STA   WINDCOLR
            JSR   GETAKEY
            ORA   #64
            CMP   #"W"
            BNE   NOWORD
ERASWORD    JSR   ERA1
            JSR   WRIGHT
            JMP   ERA2
NOWORD      CMP   #"S"
            BNE   UNSENT
ERASENT     JSR   ERA1
            JSR   SRIGHT
            JMP   ERA2
UNSENT      CMP   #"P"
            BNE   NOPAR
            JSR   ERA1
            JSR   PARIGHT
            JMP   ERA2
NOPAR       JSR   CHECK
            JMP   SYSMSG
ERA1        LDA   CURR
            STA   DESTL
            STA   SAVCURR
            LDA   CURR+1
            STA   DESTH
            STA   SAVCURR+1
            RTS
ERA2        SEC
            LDA   CURR
            STA   FROML
            SBC   SAVCURR
            STA   GOBLEN
```

Beginner's Corner: "Up, up, and away"

Pg. 71 of the *C64 User's Guide*, revisited

Sprites, then and now

Foenix Rising (reprint) July, 2022

Introduction to this column: In this first issue of *Beginner's Corner*, we will be leveraging BASIC816, simply because it's accessible to ~70% of the Foenix platforms that are in user's hands today. Next month, we will transition briefly to 65816 assembly, in order to tie some of the concepts covered (and a few others) to the good work that PJW has invested in his 65816 videos.

Longer term, we will land on something platform independent; may be the C Language or possibly a new multi-platform version of BASIC. With multiple efforts in the works, we expect to know more by the end of the Northern Hemisphere summer.

Personally, I'm a big believer in high-level (especially interpreted) languages for educational purposes because they promote experimentation and learning without requiring complicated tool chain knowledge, mastery of a language, or fear of losing work when things go 'black'. BASIC was, is, and until further notice, will continue to be, a reasonable tool for beginners.

If *retro* = nostalgia, you'll be hard pressed to find something as nostalgic as the famous 'Commodore Balloon Sprite' demo, which was (for many) a first taste of the power of Sprite Graphics, and one of many features that put the C64 on the map.

Yes, Atari had Player/Missile graphics and of course, Apple (part of the 77' Trinity) was first to feature a color [but non-linear / difficult to manipulate] bit mapped display; but Commodore took a fair amount of heavy lifting off the shoulders of would-be developers (and the 6510 processor) by creating a custom ASIC to deal with the headaches of managing and manipulating layers of movable objects, controllable via a relatively easy to understand register scheme.

Page 71 of the C64 User's Guide included a short BASIC program that out of the box, provided near instant gratification with just 5 minutes of typing. The curious among us modified the code to change the color, speed, size, direction, and in some cases, the actual image from the data statement defined hot air balloon you see on this page.

Upon typing `RUN` and pressing <RETURN>, something wonderful happened, and I'm not referring to the balloon. It was a wondrous new sensation followed by a realization: "I did that !! I can program a computer !!".

```
1       REM UP, UP, AND AWAY
5       PRINT "{CLR/HOME}"
10      V = 53248 : REM START OF DISPLAY CHIP
11      V + 21,4 : REM ENABLE SPRITE 2
12      POKE 2042,13 : REM SPRITE 2 DATA FROM BLOCK 13
20      FOR N = 0 TO 62 : READ Q : POKE 832 + N, Q : NEXT
30      FOR X = 0 TO 200
40      POKE V + 4, X : REM UPDATE X COORDINATES
50      POKE V + 5, X : REM UPDATE Y COORDINATES
60      NEXT X
70      GOTO 30
200     DATA 0,127,0,1,255,192,3,255,224,3,231,224
210     DATA 7,217,240,7,223,240,7,217,240,3,231,224
220     DATA 3,255,224,3,255,224,2,255,160,1,127,64
230     DATA 1,62,64,0,156,128,0,156,128,0,73,0,0,73,0
240     DATA 0,62,0,0,62,0,0,62,0,0,28,0
```

Gotta love* the vintage color palette !!
(320 x 200 resolution - for ref/scale, each character is 8 x 8 pixels)

The Commodore 64 example was made possible thanks to the C64's VIC-II chip which supported 24 by 21 pixel single-color Sprites. Each Sprite required 63 bytes (or **504 bits**) of memory. 'Multi-color' Sprites reduced the horizontal resolution to 12 (double-wide) pixels and the same 21 vertical pixel resolution, however only two additional colors could be chosen and were shared across all 8 Sprites. Things are different now.

---

* the other thing I '*love*' about the code above is the line numbering: 1, 5, 10, 11, 12 ... COME ON, COMMODORE !!!

Back Page - Foenix Rising Issue #2 takes a look at COMPUTE Magazine Issue 2

Foenix Rising (reprint) Aug., 2022

Compute began as a Newsletter, exclusively for the Commodore PET

In this first of two parts, we'll take a look at the origins of what would become a flagship computer magazine and touch on a few products, distractions, and technologies that fostered the growth of an industry

I was at the right place at the right time 'recently' when a Minnesota based retro enthusiast offered up "old issues" of his vintage computer magazine collection for free. He wasn't keen on cataloging them ahead of shipping, or entertaining questions about individual issues, but the first to take them all and pay shipping, got them. He was clear, however, that he was keeping issue #1 of each publication. I may have paid $30 for shipping of a 12" x 12" box, the details are foggy. This *recent* event was 15 years ago.

25 years prior to that, I was fortunate to have access to some of these publications when they were new, and like many across those years, the usefulness of having a monthly journal deliver a continuous supply of articles chock full of technical detail, tutorials, and type-in programs, could not be matched. Of course, there were ads; tons and tons of ads which today, provide fodder for articles in their own right (such as last month's *Back Page* on "Skyles 1541 Flash!"), or amazement ("I can't believe such a product existed, wow!"), and also amusement ("what?! People paid that much for …"). It was a different time and place then exists today, or has existed since the birth of the modern Internet.

There was a time when two streams of content coexisted peacefully; a healthy print publishing industry, and an emerging pre-commerce (largely, University and Government funded) Internet, which offered managed Gopher, Archie, [and most of all] Usenet News services to those of us with access to Unix or VMS based school or work computers.

SGML (structured generalized markup language) hadn't yet led to a definition of HTML, and NeXT Computer would not have been invented for another 6-8 years and as such, Sir Berners-Lee was nowhere near prototyping his first HTTP Web Server.

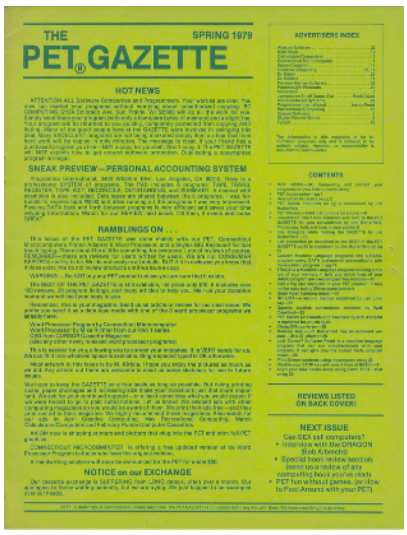
You see, in the mid-to-late 80's and early 90s, Internet content was actually 'free' and printed material was worth the cost of subscription. BBS'es weren't bad either, and other options were emerging as well (such as Compuserve, Fidonet, Qlink, and moderated forums such as "The Well" aka *The Whole Earth 'Lectronic Link*).

It is ironic that just about all of the vintage print based material is catalogued and available for free today, meanwhile, you cannot even click on a recently published news article (or YouTube vid) of any kind without hitting a paywall or being force-fed "news" about how a single mother of 3 discovered an indispensable teeth-whitening trick or "32 child celebs from the 90s - you won't believe what they look like today !!". Ok, enough of that… I do apologize.

The point of this article is a recollection of better times, and Compute Magazine joined TPUG, Transactor and others to report and disseminate news and innovation from a burgeoning industry, the first of its kind.

The beginning - not the beginning

In doing research for this article, I came to learn that even issue #1 of Compute, was not issue #1. Compute was the 2nd iteration of a publication that began through the efforts of one Len Lindsay, who in April of 1978, decided to put his College Newsletter Editing skills to work; to aggregate Commodore PET resources and information into a folded 5.5" x 8.5" monthly titled the "PET GAZETTE".


THE PET GAZETTE
SPRING 1979
HOT NEWS
SNEAK PREVIEW—PERSONAL ACCOUNTING SYSTEM
RAMBLINGS ON . . .
NOTICE on our EXCHANGE

Len Lindsay's publication just prior to transition to 'COMPUTE.'

Len's first four issues were indeed published monthly, but success turned to growth which grew to become a burden, and it forced a seasonal issue frequency and ultimately, pushed Len to turn the publication over to "Small System Services", who had reached out to him with interest right about the time that he was considering calling it quits. Len went on to contribute to Compute as an author and an editor; You can read the 'early' days of Len's story here (from Compute Issue #1, pg. 65).

A very quick look at COMPUTE Issue #1

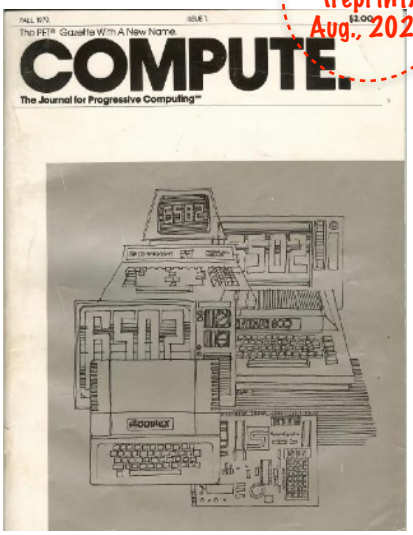
The PET GAZETTE closed with a "Super-issue", so it was only fitting that Compute begin with a seasonal (rather than monthly) "Fall of 1979" blockbuster of its own.

Issue #1 of Compute was composed of 108 pages and contained 46 articles, 10 of which were product reviews plus a group of 4 passages representing Part 1 of Len's Word Processor run-down; There was also a hodgepodge of topics including some educationally focused, general 6502 assembly language, a few articles on the BASIC language, one on printing 'pin-feed' labels, and an intro to a computer science level look at sorting algorithms, to name several.

Issue #1 of Compute broke from the Gazette's Commodore PET *only* format, but PET related topics still dominated the issue and it attracted a growing list of contributors such as Jim Butterfield, and numerous others.

Two of the more interesting articles to me (because of what came prior and what was to occur in the next 10 years) were Len Lindsay's "The Evolution of a Magazine" retrospective, and the very much forward-looking "Publisher's Remarks" editorial, which included Editor Robert Lock's welcome notice, and framed his hopes for the new publication.

The two-part "Sorting Sorts" is probably my single favorite article because it was rare in the early days of this publication to see advanced software development or data structure topics interspersed with consumer and hobbyist content. In these early years, enthusiasts with SBCs were incorporating circuits that were on-par with industry quality integration (probably because many worked in electrical engineering in the first place), but software coming from 8-bit systems was just getting started.



COMPUTE. Issue #1 - note the reference to PET Gazette

COMPUTE Issue #2 - size, shape, and distractions

The remainder of this article discusses Issue #2 of "COMPUTE. - The Journal for Progressive Computing™" as published by Small System Services in Greensboro North Carolina, USA. We will continue on this topic in Part 2 of Foenix Rising #3.

Issue #2 was published as the "January/February 1980" issue; at least, it was an issue said to represent the first two calendar months of what was to be a monumental decade for the magazine and the industry.

Robert Lock, editor, quoted the time to publish an issue as a 4-5 month endeavor so it's clear to see that with the exception of some last minute product release bulletins, the publishing workflow of the day didn't square well with the turnaround time offered by printers of this small but growing quantity.

The newsstand cost of Compute was $2.00 and Issue #2 was 116 pages long including a 16 page buyers guide coined the "After Christmas Buyers Guide" (not to be confused with Issue #1's "Christmas Buyers Guide" which was nearly identical :)

COMPUTE. Issue #2

Loosely, the issue had a preamble containing a table of contents, editor notes, and miscellany, with the remainder of the magazine split into two parts, bisected by the aforementioned buyers guide. Said guide looked as if it was printed inexpensive copy paper, the type you used in your home printer when you're out of white paper (it was light blue).

The buyers guide was more of a disorganized selection of ads than anything intentionally structured, but looked at individually, there were some gems within the guide and throughout the magazine; one example was the Eastern House Software's "PET Breadboard Kit".

For $39.95 USD (postage included), the buyer gets a 'kit' composed of a breadboard and mounting plate, a 36" ribbon cable with 24 pin DIP plug, a connector for the PET's IEEE / User port, and and a clip lead, for power. "*Now you can build those circuits you see in magazines easily and quickly!*", the advert urged.

Later in the same issue, Eastern House Software advertised some *actual* software: the MAE (PET Macro Assembler, for $169.95); this was the predecessor to the two-pass Commodore branded "Commodore 64 Macro Assembler", one that I relied heavily upon a few years later. It's fun to connect the dots backwards to see the origins of where some of these companies originated.

One of the first pieces of reader feedback was from a guy named Dennis C. Hayes (yes, *that* D. C. Hayes). He was writing in response to an issue #1 review of one of his products. Really, he was writing to plug his other products but also to complain (mildly) about the fact that his $395 Apple Micromodem™ product and accessories were reviewed "without being noted as trademarked products" (tsk tsk), but also that the magazine did not refer to his outfit by its proper registered name, "D. C. Hayes and Associates, Inc.".

A vintage detour - the rise and fall of D.C. Hayes and their Smartmodem^{tm}

fig. 30a

Of course, from proprietary beginnings (Apple specific ROM encoded signaling of the Micromodem), Hayes and his associates would go on and revolutionize the modem industry with the release of the Smartmodem^{tm} (1981) and its "ATtention" command set method of controlling it, which obviated the need for complex and proprietary interfacing in favor of simple text commands that were interpreted by the device over RS-232 serial lines.

The Smartmodem was expensive, but the "AT" protocol made the Hayes usable to Commodore, Atari, Apple, dumb terminals or Unix hosts without fuss. And this feature-set was ripped off by every manufacturer including Commodore with their 1200 baud modem.

This facet of the peripheral industry grew massively right through the PC clone revolution, though Hayes was wiped off the map after a few poorly timed bets on ISDN and a fierce battle with US Robotics, not to mention a growing field of also-ran, low-cost manufacturers including one with an ominous name: "Prometheus". Hayes never saw them coming; had they been on the lookout for the "Titan who was chained and tortured by Zeus for stealing fire from heaven and giving it to humankind", we wouldn't be having this conversation. :) Again, I apologize.

Hayes also botched licensing opportunities and mismanaged the handling of patents and lawsuits, which makes Dennis Hayes letter to Compute even more ironic.

I still have my *Smartmodem 1200* and it still works to this day. It proudly powered my Commodore 64 based BBS for a few years when I was in Berkeley; I still have not located the sister product, the *Hayes Stack Chronograph* (see figure 30a). It could be that Hayes did not sell many; the Chronograph was an expensive solution looking for a problem.

As Foenix owners are likely aware, all Foenix computers ship with an RTC (real time clock) circuit; feed your Foenix a CR2032 battery and you've essentially reproduced what Hayes once sold for $249 USD (about $800 in 2022 dollars), but directly memory mapped as binary coded decimal.

The Hayes Chronograph on the other hand, would communicate over 300 and 1200 baud serial lines with a host computer to exchange 'real time' chrono data from this single purpose product. Once configured, you could ask it "ATRT" and retrieve "123015P" (12:30pm and 15 seconds) over the serial port. You could also set an alarm and have the chrono raise the ring-indicator (pin 22) high when the set time had been triggered.

I suppose there were practical uses for it, but there were better things to spend money on in the 80s. Such as a computer. You might create a 'time card' application for a small business, coordinate transmission of file transfers for off-peak hours to save toll charges, or guarantee that logged events (such as user logons) were recorded according to an out-of-band, but accurate and controlled time source. The industry and the pages of Compute, BYTE, and other magazines were full of products like the Chronograph.

COMPUTE had articles too

The first half of Issue #2 contained just three or four general (multi-platform applicable) articles, and many that were PET specific. More than likely, the editors had to push content around in an attempt to have the *Buyers Guide* act as a separator between the general magazine and the Gazette portion, the latter of which was indeed platform focused. Compute did something similar in Issue #1 as well, but quickly gave up on the idea eventually.

In the early months of the publication, they struggled to get their hands on Apple and Atari content, logging 2.5 pages for the Apple and 7.5 pages for Atari in this issue (though the first 4.5 pgs was a comparison of Atari and PET BASICs).

Compute would not have this problem for long; by mid way through their first year, the quality and quantity of articles improved dramatically and they likely had a backlog of cutting room floor content to choose from. They eventually started paying authors by the page ($50), and by January of 1981, had moved to a monthly format.

Here is a list of of articles from the first 'half' of Issue #2, a few of which are tagged (x) for discussion below.

	Sorting Sorts, Part 2 by Rick and Belinda Hulon	PET*	3.5 pgs
	Memory Partition of Basic Workspace by Harvey B. Herman	PET	2 pgs
	Home Accounting, Plus An Easier Method of Saving Data by Robert W. Baker	PET	4 pgs
	Word Processing. A User Manual of Reviews - Part 2 by Len Lindsay	PET	5 pgs
	Book Review: 6502 Assembly Language Programming by Jim Butterfield	general 6502	1 pg
(x)	Machine Language Versus BASIC: Prime Number Generation / AIM 65 by Marvin L De Jong	AIM 65*	2 pgs
	BASIC Memory Map (Page 0): Aim, Kim, Sym, PET, Apple compiled by Jim Butterfield	Multiple	1 pg
(x)	Ramblin' by Roy O'Brien	PET	1 pg
	The Learning Lab by Marlene Pratto	PET*	1 pg
	Micros and the Handicapped by The Delmarva Computer Club	general	1 pg

* one or more of the examples referenced are applicable or convertible to other platforms

A closer look at Marvin L De Jong's Prime Number Generation article

Dr. Marvin De Jong was a Physics professor at The School of the Ozarks in Pt. Lookout, Missouri and authored a series of books on Computer Science, Physics, and Mathematics in addition to a handful of articles for various publications.

In this article, Dr. Marvin provides commented 6502 source code and an explanation supporting an algorithm to calculate prime numbers of the form 2^N-1.

The code provided was designed for an AIM 65, but the platform specific portion is limited to a single JSR to print the ASCII character (number) from the accumulator to the screen; this code should easily port to a Commodore 64, 1 MHz. reference system using CHROUT $FFD2 and to a C256U Foenix system using PUTC $00:1018.

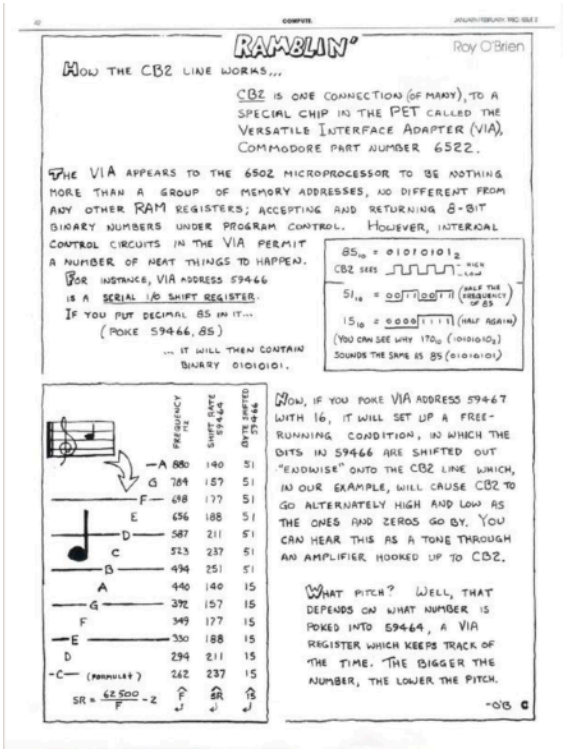
The author introduces the subject recalling that one of his students was searching for 'perfect numbers' and wrote a BASIC program for an Apple that took 11 hrs to produce a desired result. In response, Dr. De Jong rewrote the program in assembly language and executed it on the AIM 65, requiring only 11 *minutes* of run time.

Commented source is provided for the 86 instruction program; based on the text, the program requires an additional 3K of working space in order to store the resulting number in BCD. I won't pretend to be able to understand how he goes about generating said large prime numbers, but you can :). What I will do is assemble it and post it on the Foenix Marketplace as part of the September update. I'll include relative timing in the notes on the Marketplace as well.

Following this (his first article published in Compute), Dr. De Jong went on to write others including:

- Experimenting With The 6551 ACIA - March, 1981
- Computer Communications Experiments - March, 1981
- A Floating-Point Binary To BCD Routine - April, 1981
- A General Purpose BCD-To-Binary Routine - October, 1981
- Plotting Polar Graphs With The Apple II - February 1982

"RAMBLIN'" by Roy O'Brien - what is this all about?!


RAMBLIN'

Roy O'Brien

HOW THE CB2 LINE WORKS...

CB2 IS ONE CONNECTION (OF MANY), TO A SPECIAL CHIP IN THE PET CALLED THE VERSATILE INTERFACE ADAPTER (VIA), COMMODORE PART NUMBER 6522.

THE VIA APPEARS TO THE 6502 MICROPROCESSOR TO BE NOTHING MORE THAN A GROUP OF MEMORY ADDRESSES, NO DIFFERENT FROM ANY OTHER RAM REGISTERS; ACCEPTING AND RETURNING 8-BIT BINARY NUMBERS UNDER PROGRAM CONTROL. HOWEVER, INTERNAL CONTROL CIRCUITS IN THE VIA PERMIT A NUMBER OF NEAT THINGS TO HAPPEN. FOR INSTANCE, VIA ADDRESS 59466 IS A SERIAL I/O SHIFT REGISTER. IF YOU PUT DECIMAL 85 IN IT... (POKE 59466,85)

... IT WILL THEN CONTAIN BINARY 01010101.

85_{10} = 01010101_2

CB2 SEES

51_{10} = 00110011 (HALF THE FREQUENCY OF 85)

15_{10} = 00001111 (HALF AGAIN)

(YOU CAN SEE WHY 170_{10} (10101010_2) SOUNDS THE SAME AS 85 (01010101)

	FREQUENCY Hz	SHIFT RATE 59466	BYTE SHIFTED 59466
A	880	140	51
G	784	157	51
F	698	177	51
E	656	188	51
D	587	211	51
C	523	237	51
B	494	251	51
A	440	140	15
G	392	157	15
F	349	177	15
E	330	188	15
D	294	211	15
C	262	237	15

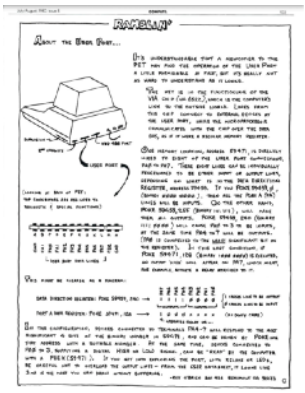
SR = 62,500/F - 2

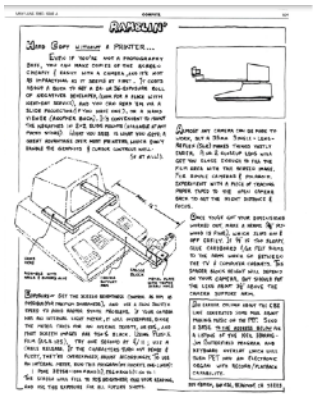
NOW, IF YOU POKE VIA ADDRESS 59467 WITH 16, IT WILL SET UP A FREE-RUNNING CONDITION, IN WHICH THE BITS IN 59466 ARE SHIFTED OUT "ENDWISE" ONTO THE CB2 LINE WHICH, IN OUR EXAMPLE, WILL CAUSE CB2 TO GO ALTERNATELY HIGH AND LOW AS THE ONES AND ZEROS GO BY. YOU CAN HEAR THIS AS A TONE THROUGH AN AMPLIFIER HOOKED UP TO CB2.

WHAT PITCH? WELL, THAT DEPENDS ON WHAT NUMBER IS POKED INTO 59464, A VIA REGISTER WHICH KEEPS TRACK OF THE TIME. THE BIGGER THE NUMBER, THE LOWER THE PITCH.

-OB

The 2nd reason this articles appeals to me is that the VIC20 (which would have basic tone generation capabilities), was not even announced at the time of this writing and while other platforms from competing manufacturers did offer audio capabilities, this method generated a fair amount of excitement for PET owners.


RAMBLIN'


RAMBLIN'

Finally, this article appeals to me because it is hand drawn; the publisher probably had no other means of easily producing the table and diagrams.

Mr. O'Brien would go on and publish more in subsequent issues; to the left is a sample of two others, published during Compute's first year. The camera idea is particularly entertaining; look for these on archive.org.

Next issue, we will pick up where we left off, examining a few more vintage ads and looking more closely at the 'Gazette' section of Compute, issue #2. Until then …