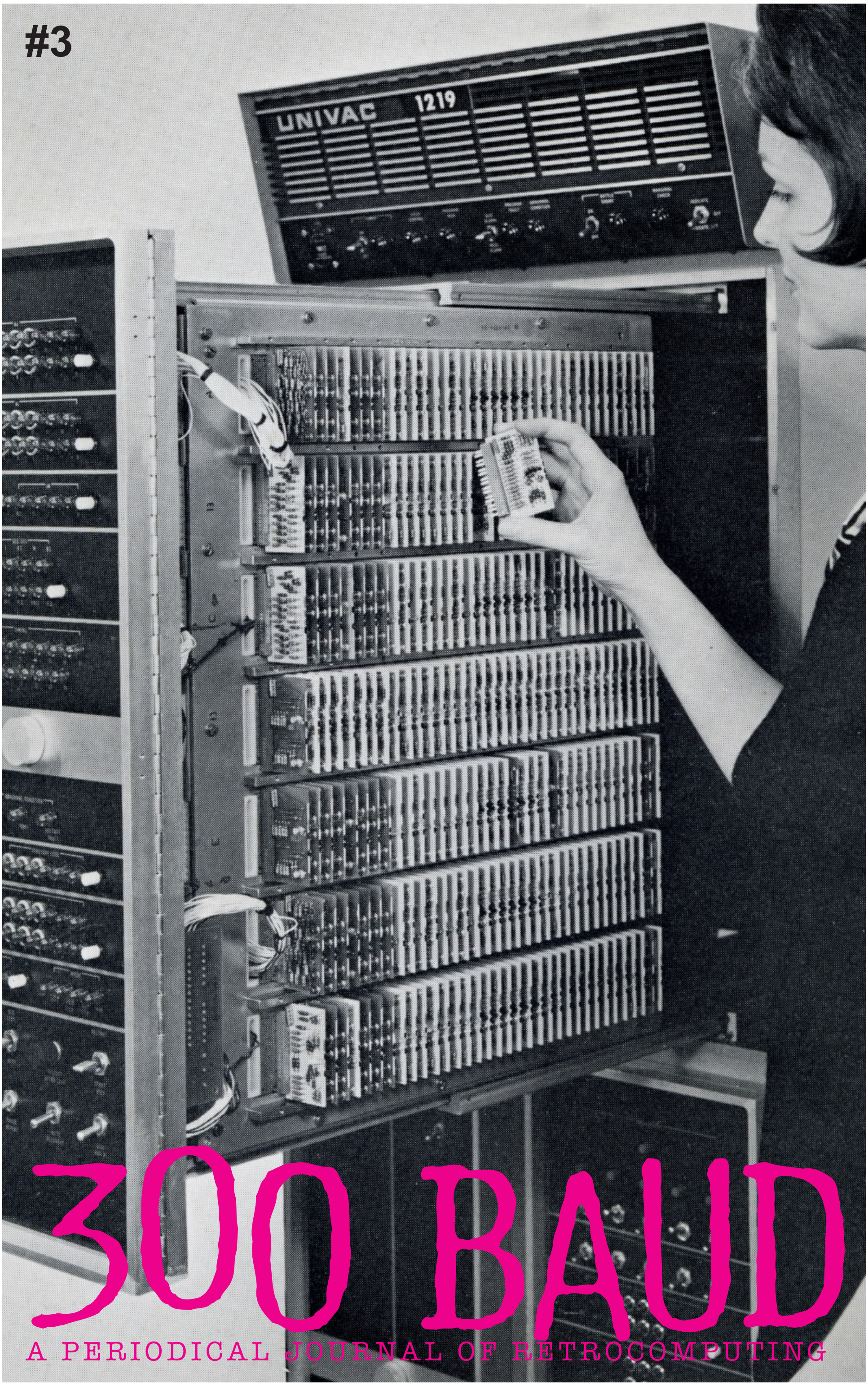
#3



300 BAUD

A PERIODICAL JOURNAL OF RETROCOMPUTING

Well, well, well! We actually did it and we find ourselves back for another issue.

Its been hard going but it is now complete and in your hands. Once again thanks to everyone that made this possible, the writers in particular who gave up their time and creative efforts. And another thanks to the people who came out of the woodwork between issues to offer a hand in other ways. All going well we will march on forwards, towards issue four, but honestly, we cant do it without you. Really.

Going back a bit to last issue, it is worth throwing out a “Congratulations!!!” to Derek, who won the Atomic Robot competition. Well done Derek for walking to the post office and being randomly chosen!

From my perspective receiving the post cards, it very was nice seeing stamps from different parts of the world. It seems I can pretty much geek out about anything. Thanks for sharing.

Anyway back to this issue. As always we have some things of interest to the newbie and to the hardcore geek. At one end of the spectrum, Gavin Picknell returns to offer us his newbie guide to using C compiling software for the 6502 and at the other, we have Paul Farrows deconstruction of the famous ZX80 flicker free space invaders game. A description of art within coding if there ever was one.

You will also notice within this issue, that we have picked up a cartoonist! Yes we now have Atticus Pamer onboard and mightily grateful for his time and efforts we are too. All going well we hope to have him slouching around the imaginary offices of 300BaudMagazine for many issues to come, well, at least for issue four!

You may also notice in this issue that there are some adverts or placements. Be calm! No money changed hands!

As we said with issue one, if we are able to help out fellow retrocomputerists with their projects and hobby sales then we are happy to do that. Ours is a small world and a leg up seems only decent.

So to state again to you all, your payment for 300BaudMagazine really does only go towards the postage and materials and if we are lucky a wee bit of savings for things like the Atomic Robot competition or the CD which came with issue one.

Rupert Murdoch and his hot bed of journalistic capitalism we ain’t.

~ Dale

VCF-GB REPORT

AN EXHIBITOR'S PERSPECTIVE

MARK WICKENS

As the cat winks at me from a cushion on the sofa, and Stevie Wonder sings ‘everything is all right’ I start penning my thoughts on the first ‘official’ Vintage Computer Festival in the UK, *ever*. Having never quite made it to VCF-E (the European VCF held in Munich every year in the Spring) and after my own DEC Legacy Event in aid of the National Museum of Computing, it seemed appropriate to show up with my DEC gear at an event geared firmly towards the masses.

VCF-GB was an event that took place over a weekend in mid June at Bletchley Park, home of the UK’s World War Two code breaking efforts and hosted by the National Museum of Computing(which is located on the same site)

The plan was to show two main bits of kit - a VAX 4000/90 and a DEC 3000/600 – and to promote HECnet and the Retrochallenge. Having turned up on Friday to set up I found myself in a corner of a marquee (one of two I was to later discover). The VAX refused to boot. Clearly it wasn’t a fan of the Southern Electricity companies’ juice. I took it apart (which during the weekend became a lesson in sound DEC engineering, not requiring anything more than one finger to take apart) and sought council from the learned around me. Consensus was that something had been jiggled on the way down, and memory was the prime candidate. I learnt the rubber trick (rubbing the gold contacts of the memory SIMMS with an eraser to remove the oxidization layer) but to no avail. Everything that was in a socket or removable was re-seated.

Saturday saw a rousing opening speech from Chris Searle (iconic presenter of the ‘80s BBC Computer Programme) and a healthy influx of visitors curious about all that was around. Highlights in my tent were a line of Dragons (numerous of which

I had a play on, including an excellent implementation of Scramble), a recreation of the MK14, Sinclair's precursor to the ZX80, and a reverse engineered Sinclair Spectrum Ferranti ULA. The 'tweeting' Sinclair Spectrums, connected to the internet using a custom PCB proved a popular distraction. Moving further afield the 'other' tent was occupied for the most part with Amiga hardware, both old (68k based) and new (PPC based) including the brand new Amiga One X1000 - it would appear that there is life in the old dog yet. Jim Austin had dipped into his extensive collection with some rare gems, including a rebuilt ASR-33 Teletype, the sound of which made the hair on the back of my neck stand proud. Saturday was finished off with a look at the innards of a Cray YMP supercomputer currently being resurrected. Whilst impressive, the power consumption of this model pales into comparison with their first super computer, the Cray-1A, a (not so cool) 115 kilowatts!

On Sunday I managed to sneak a peek in Bletchley Hall. It had been turned into an Acorn wonderland for the most part, from Atom to BBC to Archimedes. Robotic arms engaged children and other highlights included the Commodore 64 based Guitar Hero clone. Unfortunately I wasn't able to attend the many presentations, and indeed the ELO concert (which sold out in a matter of hours)

Back on my stand, I had a mixed reception from visitors for my '90s based collection. Varying from blunt 'that's not vintage, we still use those at work' (points at the DEC 3000/600 which is 16 years old) to people hoping to see a PDP or two. Clearly I need to learn up on PDP history - from 8's to 11's I get very confused as it's never been my field of interest. Happily there were also those appreciative to see some non-gaming hardware on display. One pair of enthusiasts remained perplexed as to why VMS kicked into touch a 'uname -a' request but even more curiously spent an inordinate amount of time examining the output provided by Digital UNIX.

There was a good amount of media interest – my Alphaserver 3000/800 motherboard was shamelessly used as a background to the lead in picture on the BBC news website, and I was interviewed for the BBC technology programme 'Click'. Since the event there has been several reviews posted.

After an arduous slog back home on Sunday afternoon (instructed to get home before the kids went to bed – on pain of death, it being Father's day and all) my dismay at the dead VAX 4000/90 turned to joy when it booted first time. I'll be back next year if the event becomes annual, although it may be as a visitor – I'd have liked much more time to play...

(Written using ALL-IN-1 on the VAX. What is most noticeable about using an old word processor is the omissions from the spell checker – it's a real eye opener, for example 'internet' is an unknown word...)

STARTING OUT WITH CC65

BY GAVIN PICKNELL

I love my 8-bit Apple IIe. Some of the appeal is the flexibility of the platform. If you can dream it and have enough skill, chances are you can make it happen. Myself, I can dream it, but tend to be a bit lacking in the skill area. In the retro computing world you're likely to find enthusiasts that have more than their fair share of brain power. Enthusiasts who understand every aspect of their favourite system down to what cycles do what on the bus. I am not one of those enthusiasts. Sure, I know a bit about computers and have been around them for years but I am more of an end user. Are there many end users of retro platforms left? I suspect at least one or two and the funny thing is an end user of a retro system is quite a different thing to an end user of a modern system. On retro systems end users were expected to create. BASIC wasn't built into the majority of platforms for nothing. I am no programmer but have been known to write the odd BASIC program as I would guess is the case for anyone using retro platforms today. But what do you do when you want more and are a self-proclaimed end user? Assembly language has a step learning curve and is very specific to the platform you are working with, no matter what system you use. Also, although some 8-bit home computers had other high level languages available for them (which tended to abstract the underlying system from the program to some degree), they are long out of date and unsupported – or are they? Enter CC65.

CC65 is "a freeware C compiler for 6502 based systems" that is actively maintained and developed, right now, today, in this crazy future we live in (2010 in case you missed it). It surely has a wide appeal given it supports a number of 6502 CPU based targets such as the Apple II and IIe (my favourite), 8-bit Atari's, Commodore's C16, C64, C128, PET, Plus/4, VIC 20 computers and some others I've never even heard of!

C is a programming language that's been around since at least the end of the 70's, and in my simplistic view of the world sits somewhere between BASIC and Assembly language in terms of power, speed, flexibility and difficulty to learn (BASIC being easy to learn, but down the ladder in terms of power, speed and flexibility – Assembly of course being at the opposite end of the scale).

The value of having a modern C development environment that can target so many retro platforms should not be underestimated. These days someone developing a program for their favourite system might consider porting it to another platform when using CC65 as it is probably going to be a lot less work than if they had used a programming environment native to their platform.

Actually, at this point I'd like to take a moment. You know, there are some really clever people in the world, and I'd put those generous souls who develop and maintain CC65, (and specifically for the apple world Applewin and Ciderpress – more about them later) in that group. These tools are designed for technology long past its prime and yet there's a dedicated bunch of skilled people that continue to develop, maintain and support it. They clearly have a personal interest in doing so, but no obligation to share it with the rest of us – but they do, so thanks to you all – it makes playing with retro computers that much more enjoyable for us end users.

CC65 is well documented. The documentation covers the theory of how the compiler works and describes the use and interaction of all the components. It also includes information specific to each of the target systems it supports, provides a helpful tutorial on building a sample program (hello world) and includes a number of example programs. The documentation doesn't really cover the C language itself, this is something you will have to investigate separately, however with the provided examples and some googling (there are some advantages today over the technology of 25 years ago) I'm sure you'll be producing new programs for our old systems before you know it.

To prove to myself how easy it is (easy being a relative term, my assumption being that if I can do it, so can you) I decided to build "hello world" for my favourite target – the Apple IIe.

My objectives are:

- Use my Windows XP PC as the development platform
- Use the Applewin emulator, emulating an Enhanced Apple IIe, running on my XP machine, as the target
- Create a standalone program for use under Prodos

Now, I know this is somewhat cheating, using an emulator as the target, however, in the modern world it I find it easier to use emulation when doing creation – once everything is ironed out, oh the joy of transferring it to and running it from your real hardware. There's nothing better.

To achieve this I will need:

- CC65 (http://www.cc65.org/) – This is the C development environment and includes the sample "hello world" program
- AppleWin (http://applewin.berlios.de/) – This is an Apple II emulator
- Ciderpress (http://ciderpress.sourceforge.net/) – This is a tool for working with emulator disk images
- A Prodos system disk for use in Applewin (http://apple2.org.za/gswv/a2zine/System/ProDOS203.zip) This is version 2.0.3

The installation of the required packages is fairly straight forward assuming you've read the appropriate installation notes for each. The only one that was slightly daunting to me, before I looked at it, was CC65. This was simply because my lack of programming experience and my expectation that programming in anything other than basic was a black art. The install is actually very straight forward, especially if you use the Windows MSI installer package for it (there's binary distributions for Windows and Redhat linux and the source code is known to compile on a number of other systems including Mac OS X). The default installation installs with all the files you'll need to target an Apple IIe.

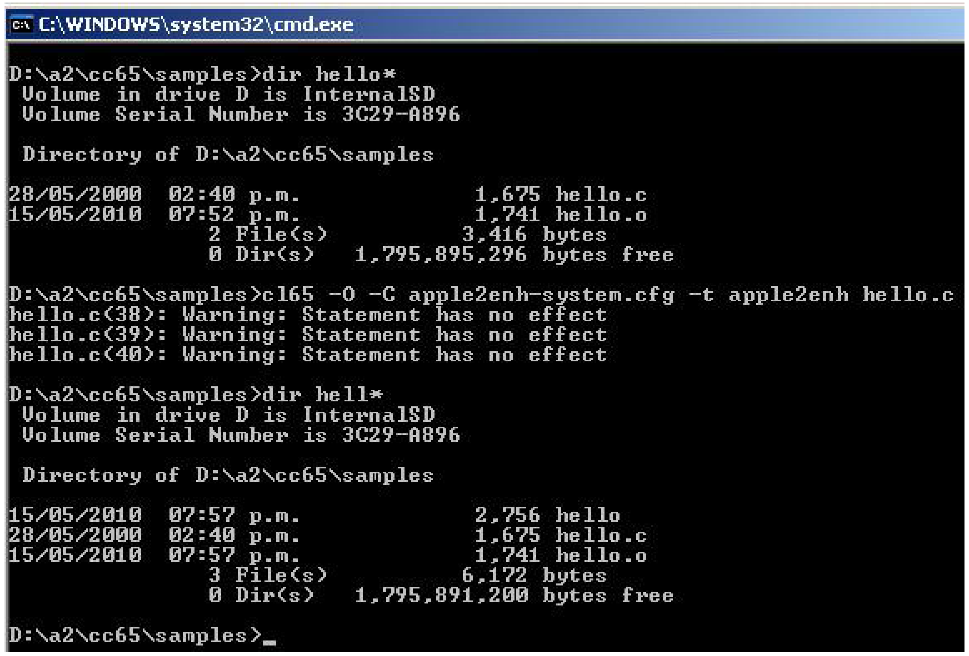
In general I recommend you read the documentation included with CC65. It will get you going in the right direction, and I'm sure for an experienced programmer would be everything you need. Also, for Apple II users don't forget to read the included Apple II specific information; I'm sure this will save you some pain.

I read through the entire thing, didn't understand it all, but it gave me a feel for what was going on. Luckily most of the complexity of the system is hidden when using the "cl65" utility and this is also well explained in CC65 docs.

Heres what worked for me (with the above objectives in mind).

Step 1 – Compile Hello World with cl65

There are several different linker configurations (you've read the CC65 documentation by now right?) available for use depending on the Apple II operating system you are targeting. From my reading of the docs, I should have been ok with the default linker config, but this didn't work out so well for me with my resultant program file not running on my Apple II emulator. In the end I specifically told cl65 to use the "apple2enh-system.cfg" config file (located in the cfg directory of CC65) to make a Prodos 8 system program and this worked. I suspect the problem was that I'm an end user.



```
C:\WINDOWS\system32\cmd.exe

D:\a2\cc65\samples>dir hello*
 Volume in drive D is InternalSD
 Volume Serial Number is 3C29-A896

 Directory of D:\a2\cc65\samples

28/05/2000  02:40 p.m.             1,675 hello.c
15/05/2010  07:52 p.m.             1,741 hello.o
               2 File(s)          3,416 bytes
               0 Dir(s)   1,795,895,296 bytes free

D:\a2\cc65\samples>cl65 -O -C apple2enh-system.cfg -t apple2enh hello.c
hello.c(38): Warning: Statement has no effect
hello.c(39): Warning: Statement has no effect
hello.c(40): Warning: Statement has no effect

D:\a2\cc65\samples>dir hell*
 Volume in drive D is InternalSD
 Volume Serial Number is 3C29-A896

 Directory of D:\a2\cc65\samples

15/05/2010  07:57 p.m.             2,756 hello
28/05/2000  02:40 p.m.             1,675 hello.c
15/05/2010  07:57 p.m.             1,741 hello.o
               3 File(s)          6,172 bytes
               0 Dir(s)   1,795,891,200 bytes free

D:\a2\cc65\samples>_
```

Figure 1 – We have complied hello (the hello world program)

The command line for this was: cl65 -O -C apple2enh-system.cfg -t apple2enh hello.c

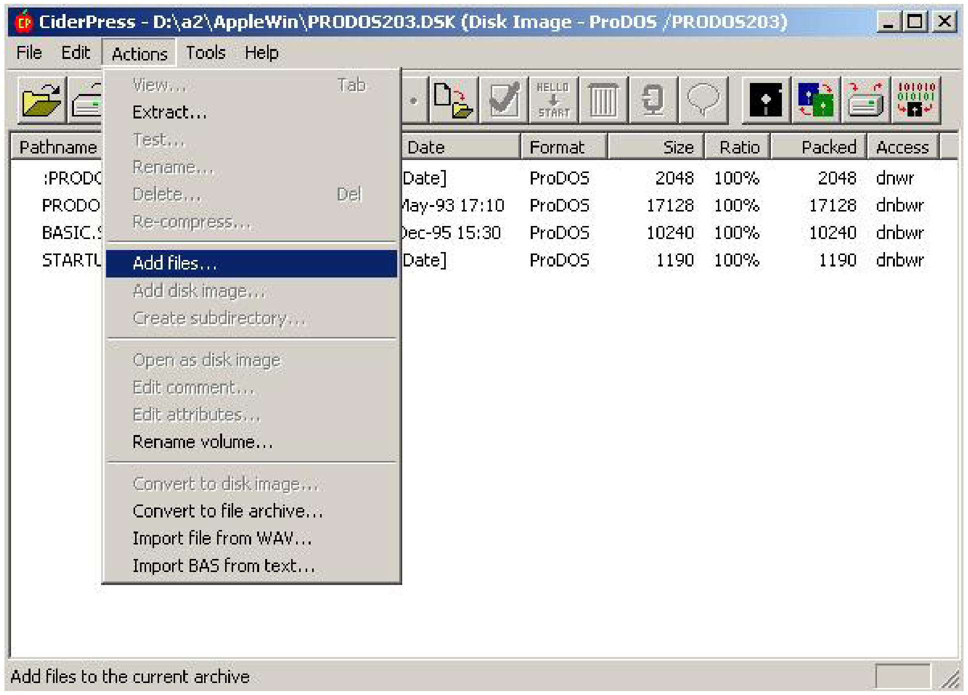
Note the "-C" option to specify the linker config file. Now, we are essentially done at this point from a CC65 point of view. We have successfully complied our "hello world" program. All that remains is to get it onto our target platform.

As can be seen from the output in Figure 1, we now have a file called "hello".

Step 2 - Get our "hello" file into our Prodos DSK image

With our program compiled into a binary, we now need to get it into a DSK image in order to be able to load it up inside Applewin and run it. The original tutorial included with CC65 describes how to use AppleCommander for this purpose but I will do it with CiderPress instead. This is because I'm more familiar it and have used it in the past for manipulating disk images with great success. When we transfer our compiled binary into the DSK image, we need to ensure that the file type is set correctly in order for Prodos to even want to load and run our binary AND we need to ensure that the programs load address is set correctly. In Prodos the load address of a program is stored in the programs directory entry (we'll see this soon), not inside the program itself (as with Dos 3.3). Having an incorrect file type or load address set will result in the program failing to run.

To transfer the program to our Prodos DSK image, launch CiderPress. Once Ciderpress is up, select "open" from the file menu .



CiderPress - D:\a2\AppleWin\PRODOS203.DSK (Disk Image - ProDOS /PRODOS203)

Pathname	Date	Format	Size	Ratio	Packed	Access
:PRODO	Date]	ProDOS	2048	100%	2048	dnwr
PRODO	ay-93 17:10	ProDOS	17128	100%	17128	dnbwr
BASIC.S	ec-95 15:30	ProDOS	10240	100%	10240	dnbwr
STARTU	Date]	ProDOS	1190	100%	1190	dnbwr

Add files to the current archive

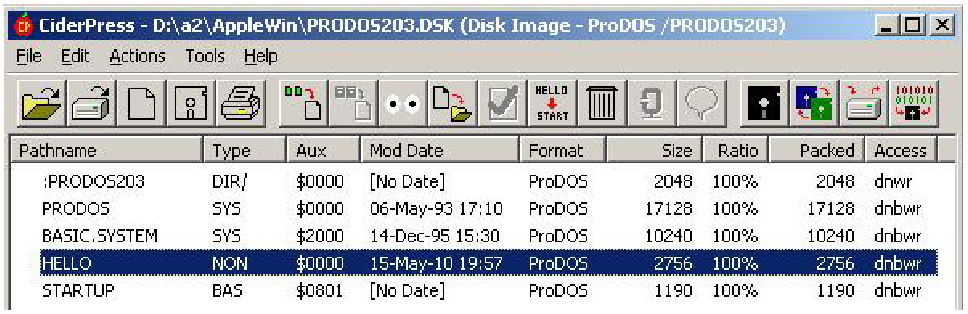
Figure 2

Browse to and select the Prodos DSK image.

Once opened you should see a listing of all the files and directories within the DSK. We need to add “hello”. To do this, select the “Add files...” option from the “Action” menu (see figure 2).

Now use the resulting dialogue box to browse for and select the “hello” file we compiled earlier. It is safe to leave all the options within the dialogue at their default settings.

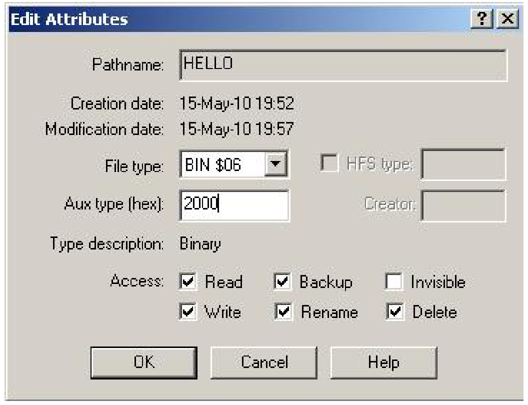
“hello” should now be added to the DSK image (see figure 3).


CiderPress - D:\a2\AppleWin\PRODOS203.DSK (Disk Image - ProDOS /PRODOS203)
File Edit Actions Tools Help

Pathname	Type	Aux	Mod Date	Format	Size	Ratio	Packed	Access
:PRODOS203	DIR/	$0000	[No Date]	ProDOS	2048	100%	2048	dnwr
PRODOS	SYS	$0000	06-May-93 17:10	ProDOS	17128	100%	17128	dnbwr
BASIC.SYSTEM	SYS	$2000	14-Dec-95 15:30	ProDOS	10240	100%	10240	dnbwr
HELLO	NON	$0000	15-May-10 19:57	ProDOS	2756	100%	2756	dnbwr
STARTUP	BAS	$0801	[No Date]	ProDOS	1190	100%	1190	dnbwr

Figure 3

As mentioned earlier we need to ensure that Prodos knows what to do with this file by ensuring that its directory entry is set correctly. The file “Type” attribute describes the type of file this is, which in this case is a binary executable (BIN). If the file type represents an executable, then the file “Aux” attribute represents the programs load address, which for this type of file needs to be $2000. Notice however that the file and aux attributes of our “hello” program are incorrect (because Windows doesn’t understand Prodos files). These attributes need to be modified to “BIN” (which has a HEX representation of $06 in Prodos) and 2000 respectively. Fortunately for us, this is very easy to do in Ciderpress. From the “Action” menu, select the “Edit attributes...” option and in the resultant dialogue box, set File type to “BIN $06” and Aux Type to “2000” and click OK (see figure 4).

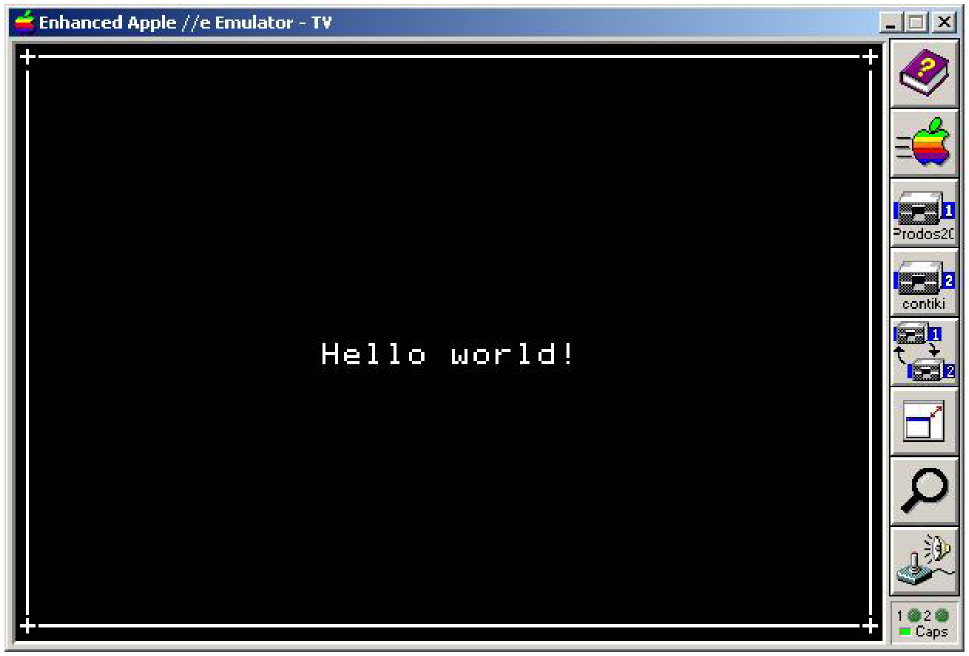


Pathname: HELLO
Creation date: 15-May-10 19:52
Modification date: 15-May-10 19:57
File type: BIN $06
☐ HFS type:
Aux type (hex): 2000
Creator:
Type description: Binary
Access: ☑ Read ☑ Backup ☐ Invisible
☑ Write ☑ Rename ☑ Delete

Figure 4

Now for the moment of truth. We should be able to launch AppleWin, boot from our Prodos DSK image and BRUN HELLO. On my PC, DSK files are associated with AppleWin so double clicking on the Prodos DSK image will launch AppleWin, place the DSK in the first drive and start the emulator. If this is not the case on your system, simply launch Applewin and insert the Prodos DSK image into the first drive, then start emulation.

If all goes well you should be able to drop to a Prodos command prompt and BRUN HELLO (see Figure 5).



```
Hello world!
```

Figure 5

You are now (I hope!) on your way to developing loads of great programs in C for your 6502 CPU based system – just make sure you port what ever you make to the Apple II!

REANIMATING THE SINCLAIR ZX80

by Paul Farrow

3D Monster Maze, Jet Pac, Jet Set Willy, Knight Lore, Psion 3D Chess – These are just a few of the classic games developed for the range of computers created by Sinclair Research and which set new standards in graphics or game play in the early 1980s. Each game has now become synonymous with the computer that it was originally created for, be it the ZX81, Spectrum or QL. There is however one member of the Sinclair computer family not represented in this list – The ZX80.

It is best remembered for its inability to produce a continuous display rather than for any genre defining game, and when the ZX81 was launched the ZX80 simply flickered into obscurity. Contrary to popular belief a sizeable range of software was produced for the ZX80, quite often sold as printouts rather than on cassette. The relatively few ZX80s sold (only about 50,000 according to Sinclair adverts) means that the quantity of original commercial software produced for the machine was comparatively low and is therefore scarce to find nowadays. The nature of these programs also tended to be fairly basic and so not very memorable, but one game which does stand out and which has in recent years become well known for the ZX80 is a continuous-display version of the arcade classic Space Invaders. In this article, I aim to explore the technique used to achieve this continuous display output and to provide an overview of the approach required to create such games for the ZX80.

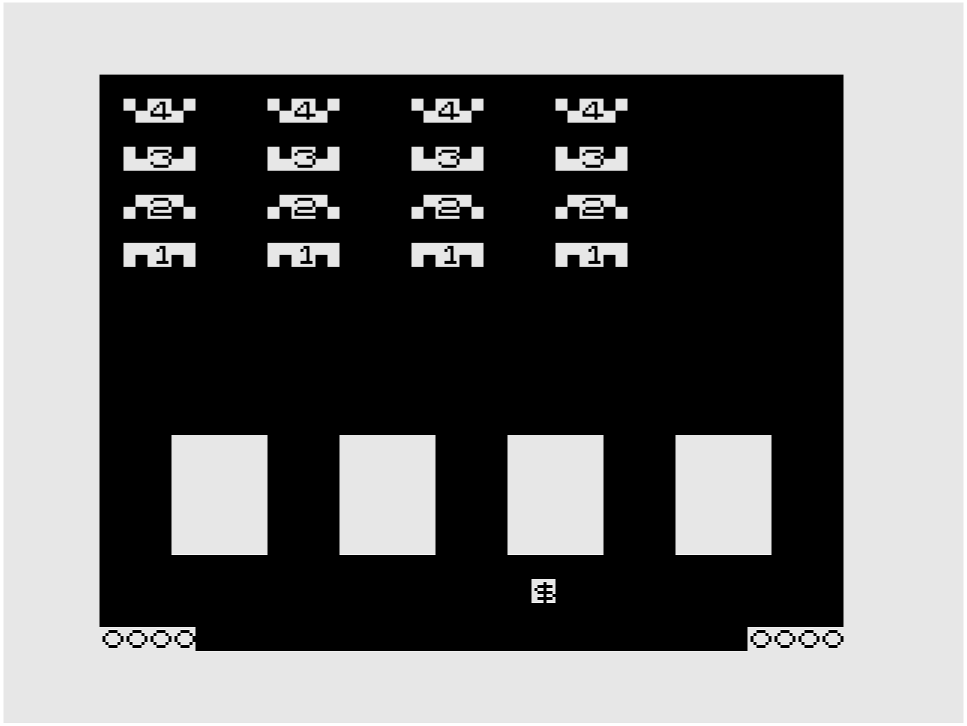
To keep the cost of the ZX80 down, Sinclair devised a mechanism for generating a television picture that was partly implemented in software. This significantly reduced the amount of hardware needed and contributed to the ZX80 being sold as the first personal computer for under 100 pounds. An implication of this cost cutting approach was that the ZX80 was not capable of displaying a television picture while it was running a user's program. The only time it could produce a television display was when it was idle waiting for the user to press a key, and this was because it had entered a precisely timed loop that generated TV frames at a regular rate. The loop produced the scan lines that formed a TV frame and then delayed until it was time to output the next frame. The delay period was put to use by using the time to scan the keyboard, and if a new key press was detected then the display loop would be exited. This resulted in the famous flicker associated with the ZX80.

To achieve a continuous flicker-free display, the timing of the fixed length delay had to be precise. Using a hardware based timer was not an option due to the desire to keep material costs down. The solution instead was to exploit the fact that the Z80 microprocessor at the heart of the ZX80 executes its instructions in finite lengths of time. Each type of instruction takes its own particular length of time to execute, and these times are openly published within the Z80 documentation. By careful selection of instructions it was possible to craft the delay to take the required length of time such that a precise TV frame rate was achieved.

It did not take software companies long to realise that it would be possible to produce a game with a flicker-free display if the game could be structured to run only in the delay slots between TV frames. The game would need to be divided into blocks of functionality, with each precisely timed to fill the slot duration, and then those blocks cycled through on each new frame. The result would be a simple form of cooperative multi-tasking between the game and the display generation mechanism. However, another problem existed. The fixed delay executed by the ZX80 between frames spanned only 400µs, which was hardly enough time for a game of any sophistication. The solution devised was to reduce the number of scan lines output per TV frame and to allocate the time saved to the game instead. This change had no visible impact on the TV picture since a television frame consists of more scan lines than are visible on the screen. These additional scan lines are used in broadcast systems to hold information such as teletext and closed caption data but since such data services were of no interest to the ZX80 it would simply output blank scan lines instead. If the number of scan lines generated was reduced to the minimum visible on screen then an additional 1ms could be freed up and used by a game instead. This would provide enough time to make flicker-free games viable.

The best known flicker-free game is Space Invaders, created by Macronics (Ken MacDonald and Ron Bissell), and it can be readily found on the Internet for download, with video clips available on You Tube. According to an interview with Ken MacDonald in Your Computer magazine (issue 1, June 1981), it was

Ron Bissell who devised the approach. Whether the technique was independently developed by others is not known, but several other companies certainly produced and sold flicker-free games for the ZX80.



Eliminating the Flicker

At first sight, the achievement of flicker-free games such as Space Invaders can seem staggering. The game must be divided into blocks of functionality, with each precisely filling the delay slot between TV frames. Every possible pathway through these blocks must be timed and balanced so that they always complete in the same length of time. Even a relatively small deviation could result in some jittering of the television picture, and so the patience and skill to achieve this perfect balance can appear amazing. So how can this feat be achieved?

The first step required is to determine the duration of the enlarged delay slot and hence the number of instructions that can be executed within it. As mentioned, each Z80 instruction takes its own particular length of time to execute, and is quoted in terms of the number of clock cycles required by the Z80 to complete them. Instruction execution times range from 4 to 23 clock cycles, and so the number that can run within a slot will depend on the type of instructions used. However, the total number of clock cycles that occur within a slot will be constant and hence it is this number that needs to be determined. It can be found by examining the delay performed by the display loop within the ZX80's ROM and extending it with the time saved for each scan line reallocated to the game. The result comes out at 4731 clock cycles. Since each instruction takes at least 4 clock cycles to execute, a maximum of 1182 instructions can fit within the enlarged delay slot. Assuming the average instruction duration is 8 clock cycles then this reduces to only 600 or so instructions.

The next step is to make a first attempt at dividing the program's functionality into roughly equal sized time blocks so that each will fit within the enlarged delay slot between TV frames. This partitioning is provisional and may need revising several times during the development of the program. As an example of the scale of functionality that can be squeezed into a time slot, the Space Invaders game is divided into only four blocks and so it takes four TV frames to cycle through all of them. This yields a nice smooth animation rate. Increasing beyond 8 blocks begins to impact the animation rate and so it is really this that sets a limit on the complexity that can be performed using the flicker-free technique.

For simplicity the program can be structured around a main loop, similar to the display loop found within the ZX80's ROM. It will consist of a sequencer, which cycles through the blocks of functionality executing one per frame, followed by calls to the display routines within the ZX80's ROM to generate the TV picture. The sequencer can be implemented using a simple vector table, which has the advantage that the dispatching process takes the same length of time to execute irrespective of the sequence entry. Each block of program functionality should handle one specific aspect of the game, and will often require a state machine to keep track of its tasks and status. The state machine behaviour could be as simple as keeping track of a missile in a shooting game, e.g. the location, direction and speed of the missile, or it could be more involved such as keeping track of a complex animation sequence. Each state should be implemented by a separate routine, with a vector table used to select which one to run upon each cycle. Again, the use of a vector table ensures that a fixed duration dispatching mechanism is achieved. So far there has been very little regard for accurate timing. It is only within the individual state routines that precise timing must now be considered.

Ideally the instructions forming a state would simply follow a single path of execution and so all that would be required to achieve perfect timing would be to ensure that any surplus time after completing the state's functionality was consumed by executing "padding" instructions. These "padding" instructions would not change the functionality of the state but would simply introduce a delay due to the time required to execute them. The Z80 microprocessor provides the NOP instruction (which stands for "No Operation") and this literally performs no functionality while taking 4 clock cycles to execute. However, the padding delay required may not always be multiples of 4 clock cycles and so other instructions must be used to achieve different delay times. For example, if a delay of 7 clock cycles was required then an instruction such as LD H,H could be used since this simply copies the contents of the H register into itself and hence the register's value is not actually changed. Most Z80 instructions will have some effect on the state of the CPU and so care must be taken to ensure that the effect does not actually change the intended functionality of the state routine. By piecing together suitable instructions, delays of almost any duration can be performed. Another factor to keep in mind is the number of bytes required to encode instructions. It is quite possible to achieve a particular delay by using a variety of different instructions yet each might be encoded using a differing number of bytes. Choosing the smallest instruction minimizes the wasted space introduced by padding instructions and helps to keep the program size down. Becoming very familiar with the range of Z80 instructions, their execution times and their encoding sizes is therefore invaluable to achieving a good, concise solution.

To produce large delays, using long sequences of padding instructions becomes impractical. For such scenarios, it is better to write delay subroutines which simply loop round doing nothing for a specified number of iterations. Such subroutines can then be called from various points within the program, thus helping to keep the program size down. The shortest delay loop possible in Z80 assembly language is achieved using the DJNZ instruction, which takes 13 clock cycles per iteration.

To call a subroutine and to return from it introduces a fixed overhead that must also be taken into account, and results in the shortest delay loop possible becoming 35 clock cycles. However, the subroutine can then be used to create delays of 13 clock cycle increments, i.e. 35, 48, 61, 74, etc. Any fine tuning on these values can be achieved by following the call to the subroutine with one or two other padding instructions.

In reality each state routine will not simply follow a single path of execution but will contain a multitude of decision points and branches. Each one of these must be balanced with padding instructions so that every possible route takes the same length of time to execute. A good approach to tackling this issue is to divide the state's functionality into multiple subroutines, with each limited in the number of decision points, and hence branches, it contains. Subroutines then become responsible for ensuring that their functionality always executes in a fixed duration no matter which route through them is taken. This allows the routines that call them to view the subroutines as fixed length delays. The number of decision points to include within a subroutine is arbitrary and will depend to some extent on the functionality being implemented. In the extreme, all subroutines could be limited to only a single decision point and so there would only be two branches to balance. Once the slowest path through a subroutine has been identified and the time it takes calculated, the other branches can be balanced accordingly. A comment should be recorded at the top of the subroutine indicating the time it takes to execute. This makes it quick to look up and reference whenever the subroutine is called.

Tracking the execution time through every route within a subroutine is best handled by liberally commenting the source code with running totals of the time taken (in clock cycles) since the start of the subroutine. This is especially important around decision points (where the code branches) and convergence points (where branches merge together). To simplify this process, each assembly language instruction should be commented with the time it takes to execute. Where a subroutine is called, the time taken to make the call and the time it takes to run the subroutine should both be recorded. It is good practise to type these comments and compute the running totals while writing the program, but note that any subsequent modification to the program will require recalculation of these totals and so even a small change can end up being fairly time consuming to make.

Any serious development of a flicker-free program is best performed on a modern PC and tested using a good quality emulator, one which preferably provides a versatile range of debugging facilities. Such debugging facilities not only allow bugs within the program to be quickly identified, but can also be used to verify the timing recorded for each subroutine. For example, the EightyOne emulator allows breakpoints to be set within a program and will display the number of clock cycles executed since the last breakpoint was hit. This makes it possible to perform checks on the timing of subroutines, although verifying every possible route through them can become impractical. Once all tests have been completed using an emulator, the program should be transferred to a real ZX80 and run to ensure that it matches the predicted behaviour.

In this article I hope to have presented a feel for the technique required to create flicker-free programs for the ZX80, along with the types of issues encountered and the strategies that can be employed to make the development process manageable and fair less onerous than might otherwise appear necessary. By careful partitioning and commenting of a program, developing for the ZX80 becomes only slightly more complex than developing for the ZX81 and so I hope more people will be encouraged to direct their programming skills towards the ZX80. This might finally see the ZX80 remembered for what it could do instead of for what it could not.

Reference links:

- www.fruitcake.plus.com
  Contains the Space Invaders game for download, along with a commented disassembly of it.
- www.chuntey.com
  Free download of the EightyOne emulator.
- www.z80.info
  Contains detailed lists of the Z80 instruction set, including byte sizes and clock cycles.

SOMETHING OLD MEETS SOMETHING NEW

by Shaun M. Wheeler

Two of the most exciting computer enhancements I've bought in the last few years came to me via the Club 100 website. They are the manly-named REX and NADSBox, respectively. Both are recently released add-ons for Tandy's nearly thirty year old Model 100/102/200 line of portable computers.

The NADSBox (designed by inventor/engineer Ken Petitt) is a modern replacement for Tandy's aging Portable Disk Drive line. Connecting via the serial port of the Model "T" computer, it enables the use of common (and inexpensive) SD cards for mass storage. It integrates seamlessly into most applications requiring a TPDD, works beautifully with Tandy's TS-DOS, and (best of all) the SD cards it uses can then be read (or written to) by any PC/Mac with an SD card reader!

That, for me, was the selling point. It meant no more fooling with null-modem transfers, slow downloads off the net, or unreliable TPDD floppy reader apps on the PC. With the 512Mb SD card included with the NADSBox, it also meant not having stacks upon stacks of floppies cluttering my office, as all of my 200+ TPDD floppies fit on the card!

The NADSBox was truly money well spent. I've had mine for over a year now, and don't know how I ever did without it.

However, as good as the NADSBox is, it isn't a hard drive. Files must still be copied onto the Model 100's RAM and used from there. If you use a number of apps, you may find yourself running low on available RAM.

This is where Option ROMs came in. Back in the day, some programs (or suites of programs) were released on ROM boards that plugged into the Option ROM socket in the back of the Model "T". The advantage to this is that software on an Option ROM was run directly off the ROM (and not copied to memory), thus saving precious RAM. Many software packages such as Tandy's TS-DOS and Microsoft's popular Multiplan spreadsheet were released as ROMs, and proved to be popular choices.

The disadvantage, however, was that the Model "T" could only hold one Option ROM at a time. When you wanted to switch ROMs, you had to "remove" the link to the ROM from memory (in BASIC), then physically remove the ROM, put the new one in, then CALL it from BASIC. Another disadvantage was the impracticality of carrying several tiny, flimsy, unprotected, easily lost ROM boards with you.

This is where our pal REX makes his entrance.

Designed by inventor/engineer Steve Adolph, REX is a Flash memory upgrade for the Model "T". It plugs into the Option ROM slot, and gives the Model "T" sixteen banks for Option ROM images, as well as up to sixteen RAM banks. REX essentially gives you up to sixteen virtual Model Ts, with the ability to switch between them on the fly, thanks to the well-written REX Manager app.

I preordered the REX from Club 100 as soon as they were announced, and within a month it too was in my mailbox. How does REX stack up?

Next to the NADSBox, REX is the most useful computer upgrade I've purchased in years. It's nothing short of phenomenal... let me tell you how.

Currently, I have four "images" on my Model 100: one for blogging, one for work, one for fun, and one "master" image. They are laid out as so:

BLOG: contains an Ultimate ROM-II image (includes T-Base, T-Word, Idea, View80, TB-RPT), a basic program to insert a byline and boilerplate into my writing, a few articles I'm working on, and my Shortwave Log database. Used primarily for writing.

WORK: Contains at times either an Ultimate ROM-II image or a Multiplan image, plus a couple of format conversion apps in BASIC. Usually used for quick & dirty sales quotes at work.

FUN: A few games, a few "serious" apps I'm playing around with (currently messing with an emulator for the HP-12C calculator), and the odd text file I downloaded to read. Used primarily for fooling around.

BACKUP: A backup of my system in its original state. Never touched.

Without REX, I would have to keep my Model 100 relatively clear of data. Writing and works-in-progress would have to be shunted to my PC any time I wanted to play a larger game or download a sizeable app or text file. If I had a lot of work to do for a customer (or managing my household finances), that might mean having to swap out my UR-II and Multiplan ROMs several times in one sitting, then transferring the results to my work PC. Without my NADSBox, it'd be even more of a chore... Definitely not constructive use of my time.

So, were the NADSBox and REX worth the money?

Yes. I wholeheartedly recommend both... they were worth every penny! The combination of the NADSBox and REX have made my life so much easier, and have kept my Model 100 a viable, secure, and eminently useful computing platform in these days of cheap Netbooks and disposable PCs.

If you're at all serious about using your Model "T" computer in a serious capacity, you'd do well to pay Club 100 a visit and pick up a NADSBox and REX.

Visit http://club100.org for more details.

300 BAUD

SERIAL INTERNET CONNECTIVITY FOR VINTAGE COMPUTERS

by Tony Fink

So, you've got that lovely beige box running again, fixed all the hardware problems, and even played about with some old familiar software. But you've run out of ideas. Now what? Well, put it online and get some use out of it! If your computer has a serial port then chances are you can use it online in either a client or server role.

Client Internet

The most common use for an old computer online is for Telnet services. Telnet can be used to connect to BBS servers, information services, or remote systems for shell access. With Telnet and a remote Linux or Unix shell, many more possibilities open up, like IRC, web browsing in text mode, server maintenance, email, and anything else you can think of that will run in the command line. For those who want a real retro experience, there are VAX/VMS systems that can be telneted into as well.

Getting set up for the internet isn't to hard to do. There are plenty of options available to get things going, but there are two crucial things that you'll need before anything else. The first is a null modem cable and the second is a terminal application for your retro computer of choice. Once you have these you'll need to get a gateway set up.

The gateway is a computer or device which attaches to the other end of the null modem cable and converts the serial data into TCP/IP packets which can travel through the internet to their destination. You can use a normal PC with an internet connection and special software installed, or a special stand alone device called a "serial device server."

If you are willing to pay the money to buy one, a physical serial device server is a good choice for getting your computer online. You don't have to keep a PC within a serial cable's reach of your retro computer at all times in order to connect to the internet, since the device is standalone. These devices are actually small embedded computers. You simply connect your retro computer to the device's serial port and the device does the rest. When you start up a terminal you can issue commands to the device through it's own command line. These devices are made by a number of different companies, the most prominent being Lantronix. Older used models are generally available on online auction sites for reduced prices. Fancy models support WiFi or can host multiple computers through one device. Some even have their own storage onboard. Each model is different though, so setting up requires that you obtain and carefully read the users manual. Since the devices are normally used in industrial or medical situations they can sometimes be confusing to set up, being aimed at experienced network administrators. However, for client use they are generally very straightforward and easy to use.

If you have a regular PC you can use it as a serial gateway as well. If your operating system supports it you can just enable serial terminal logins. This is the case for Linux and Mac OSX via such programs as 'getty' or 'screen.' If you choose this route, you simply are using your old computer as a terminal for your new computer and you'll have access to all the programs you've already installed on your PC and can access these via the command line. Of course, if your PC is online you can also access the internet from your retro computer in the same way you would via the command line on the PC.

Windows users don't have the ability to host terminal sessions through their own OS, but there are programs that can serve data through the serial port. One of these is called "BBS Server" and is available at www.jammingsignal.com. It allows outbound Telnet sessions to be made from your old computer by issuing Hayes-like commands. For example, in a Hayes modem issuing the command "ATDT 1234567" would call that phone number. When using BBS Server you just replace the phone number with an IP address or domain name and a port number and it'll connect to that server. In fact, BBS Server responds to nearly all Hayes commands in a way that closely mimics the behavior of serial modems of old, making use with vintage telecom software quite simple.

Something important to remember with vintage telecom software is the kind of terminal emulation being used. You can run your computer in "dumb terminal" mode of course, but special terminal emulations generally make screen redraws faster and allow for special text effects like bold text, drawing characters, and sometimes animation and sound. Make sure that you use a compatible terminal emulation, like the nearly universal DEC VT100. You'll also have to make sure that you properly configure your baud rate, parity, handshaking, and bit rate to match your gateway's settings.

Server Internet

Hosting your retro computer over a serial internet gateway is also pretty straightforward. Serving HTML pages isn't really something that can be easily done over a serial port, though software packages like Contiki can be made to do this. Hosting a BBS via Telnet however, is something that is far more doable with a serial internet configuration. This is because vintage BBS software is nearly always configured to "talk" through the computer's serial port using Hayes modem commands.

Some serial device servers are capable of running in a modem emulation mode, which allows them to host your BBS over an internet connection. However, most serial device servers are not designed to allow regular incoming Telnet sessions, but are designed to allow the linking of two serial device servers through proprietary protocols. Unless you are prepared to do a lot of protocol work and modification to your BBS software, or you have a serial device server designed for incoming Telnet sessions, it is not suggested that one be used for BBS hosting.

A very popular program for using a PC as a gateway for BBS hosting is TCPSER. TCPSER is cross platform, and has been reported to work on Linux, Mac OSX, and Windows. If you have multiple serial ports on the host PC, you can host a multi-node BBS as if you had multiple phone lines. If there are no available nodes on your BBS and someone telnets in, TCPSER can be configured to put them "on hold" until a node opens up, or display a message asking them to call back later. If your BBS is down for maintenance you can set TCPSER to display a message for people who Telnet in. TCPSER is a command line program, but once you know what sort of settings you want you can just write them down in a script or batch file and it will load and configure TCPSER correctly each time. The only downside to TCPSER is that there isn't very much documentation for it, so if the command line parameter hints the program itself gives you aren't enough, you might get stuck setting it up. Generally though, unless you are setting up a very complex kind of BBS, there should be no problems getting TCPSER to run.

If TCPSER seems too complex, there is a very simple alternative which although it has fewer features is much easier to get set up. BBS Server; the same program which you used to make outbound Telnet calls, can also used to host single node BBS programs as well. It has an easy graphical user interface which has places to enter in all your serial port configuration data, welcome messages, busy and away messages, and what IP address to host the server on. The BBS Server software also has some special features specific to hosting Commodore BBS with support for PETSCII as well as ASCII character interpretation. However, if you need to host multiple nodes on the BBS you'll need multiple instances of BBS Server running, unlike TCPSER which does it from one running instance. Also, BBS Server is only available for Windows. However, for basic BBS hosting it's a very simple package to get started with and is very beginner friendly. It also comes with extensive documentation that details how to get everything running.

Of course, it goes almost without saying that to host a BBS via Telnet you need to get your computer and network set up for hosting in general. Port forwarding and domain service both need to be in place in order for people on the web to reach your PC gateway and connect to your BBS.

Conclusion

Back when networks and protocols were being developed the internet was pronounced to be the 'killer app' for computing. Today with a serial port, some special software, and a little patience you can get a little more mileage and a lot more fun out of your old hardware and in doing so bring the internet to yourself and share a piece of computing past with the rest of the internet world.

If only they put as much work into the manuals as the actual computer!
HOME COMPUTER

300 BAUD

RETRO FROM THE START

by Rob Sayers

The spring of 1994 was a good time to be a geek. Apple released the first PowerPC based Macintosh and Intel released a new Pentium which had reached the unthinkable speed of 100mhz.

I was in middle school at the time and was just being exposed to computers for the first time. I often stayed after school to explore the various Macs we had in our computer lab and even wandered to the older Apple II lab for the occasional game of Oregon Trail. The computer bug bit me and I was hooked. I read every computer magazine I could get from cover to cover many times over trying to learn what I could.

Before long I reached a point where I knew more than any other student and even some teachers. There was one small problem though... I didn't have a computer of my own. I poured over catalogs and magazine ads lusting over all the machines I wished to call mine. I cleverly left ads out and dropped hints to my parents. I was the slightly more pathetic version of Ralphie from a Christmas Story. After several months of this they finally spent the money to buy me a computer. Instead of this computer being the cutting edge of 1994 technology, this beast was built in 1979: the Apple][+.

When I got it home I quickly got to work setting it up on an old folding card table in the basement with it's green monitor and two floppy drives. I then read over the stack of manuals and sifted through the caddy full of 5 1/4 diskettes that came with it. I remember not going to bed at all that night; a foreshadowing of the countless sleepless nights spent hacking that has followed.

Sure I didn't have Windows, CD-ROM, sound cards, lower-case letters or anything else all the well-off kids had on their home computers, but it didn't matter to me, I finally had a computer. I spent nearly every free moment I had using it. As soon as I got home from school I would rush downstairs, flick on the power switch and be greeted with a friendly beep following by the churning of the Disk II drive. After a quick break for dinner (mandated by mother of course) it was back to computing. I remember the headaches would get very bad in the beginning from the hours of staring at the screen. The pain did little to deter me.

Eventually I had played every game I had more times than I could count and again turned to that stack of books that came with the Apple. In particular, a book on "Applesoft Basic" caught my eye. I thumbed through it a little and finally sat down and started typing along with the exercises. At first I had little idea what I was doing. I finally started understanding the code I was typing and even reached a point where I started making my own changes to the programs in the book. I acquired some more books on Basic programming and continued to absorb the information until I started to get quite handy at this programming thing. When I took a basic programming class the following year I soon learned that I already knew the material that was taught and then some. I was impressed and thought that maybe this programming thing was for me.

That Apple was my only computer for about two years until my parents could afford to upgrade me to a Packard Bell 486, a dinosaur even when we purchased it new, but still a big upgrade. The programming bug never left me and I wrote countless lines of code in QuickBasic and eventually VisualBasic, C, C++, and so on.

Sixteen years after writing my first line of code I'm still programming, only now I get paid for it. I'm convinced that that lowly Apple][+ which was an antique years before it found its way to me was critical to me getting where I am today. The Apple II, like many computers of the day, was still tied to computing's hobbyist roots and encouraged you to explore the system, write code, pop the top and look inside. Almost all manuals I have for old 8-bit systems have information you would never dream of seeing included with consumer electronics today: programming tutorials, ROM maps, even schematics.

If fumbling around with 8-bits helped make me who I am today, what is there for today's kids? A young person getting a computer today is bound to get something running OSX or Windows, Computers today "just work" with no need to write a single line of Basic, PEEK a memory address, figure out where a card plugs in, or even a cable for that matter.

Anyone can get a computer up and going today in moments with no hacker spirit needed. I'm of course not suggesting we throw out all the refinements modern computing has given us. Rather I wonder if we lost something along the way that was worth keeping. Computers no longer teach us patience and problem solving, two skills which serve you well on or off a keyboard.

So where does that leave us? I think the computers of yesteryear were good teachers because of the reward that was offered. Learning a few lines of Basic to print your name in an infinite loop was oddly satisfying at a certain age, getting that boot floppy to load just the right settings to get that game to run was an amazing feeling, and moving jumpers around and trying INIT strings to get that modem to *finally* connect to that BBS offered a lot in terms of a payout. If we want to encourage kids today to get their hands dirty with technology we need to offer those same sort of rewards. The reason ham radio is not catching on with the younger crowd is because the only payoff is the ability to have a conversation with someone after much effort and lots of static despite the valuable insight gained from the hobby.

Linux is a big step in the right direction. In some cases just getting it setup can be difficult, and it only gets worse if you have poorly supported hardware. Still people do it for the satisfaction, the benefits of running the system, and the geek street cred that comes with running it. Despite getting more user friendly with each passing year, there is still a great hacker spirit which exists within the community. Not only that, but in the past few years the OS has become more pervasive as it is preinstalled on many netbooks and other small devices making it more accessible and available to those wanting to try it out.

As a programmer perhaps I tend to overestimate the value of programming, but I think it's a skill that can help with logic and problem solving even when you aren't writing code. Unfortunately the need for a built-in programming environment declined and finally died when Microsoft stopped including QBasic with Windows. The good news is that there are many quality free programming tools out there. One of the coolest is Scratch[1], a visual programming environment for kids which let them use a very simple and easy to grasp language to create some really cool programs and games. Going a step closing to "real" programming is Microsoft's XNA Studio: a game development environment, which lets you create games for the PC or even the Xbox 360[2]. Linux and OSX both come with a number of easy to learn scripting languages installed right out of the box including Python and Ruby. While your programming language is no longer the first thing you see when you start your machine, the past several years have seen advances in getting young people coding again.

Today's retro-nerds had the benefit of living in an age where computers helped shape us. Computers still have this ability but it requires us to dig a little. I think that we owe it to the next generation to encourage that hacker spirit by helping them find the wonder that still exists beneath the shiny gloss of today's machines.

Web Links: [1] http://scratch.mit.edu/ [2] http://creators.xna.com/

I've read some good things about the altair, John, there's even a BASIC interpereter that came out recently...
yeah, you can even connect it to a teletype or a terminal and a modem to talk to a mainframe for time sharing and stuff...
... I just wish I had known that stuff wasn't included when I ordered the darned thing!
8/26/10

THE PEOPLE YOU MEET

by Dale Goodfellow

I suppose one thing that you have to deal with when engaging in this retro lark is that often you are missing something. You may have the machine, but you lack the other bits to make it work properly, be it a drive, documentation or even just the cable to get it to talk to a tape recorder.

The other related problem that often presents is one of software. Many times you get your new, but grubby and dusty machine home to discover that while Google will throw up many pages offering the specs of the machine, few if any will actually have any software to go with it, for you to actually run.

The more popular 8-bits are of course very well represented, but you do not have to stray very far these machines to find your big Googley internet has just been reduced to one or two people who keep the machines web presence active, rather than just a few abandoned pages with the same stock photographs.

And so it was recently for me when I acquired my Nascom 1 and needed help getting it up and running and Idiscovered an internet stranger called Richard who stepped up to offer code and hi-res pictures of the board to help me repair it. And so it was with my Sharp MZ80K when the motherboard died and an internet stranger called John offered telephone support and then offered to actually try and repair it for me if I sent it to his house. And so it was for me when I was given my Sinclair ZX80.

Quite quickly when I got it, I discovered that there is precious little software out there on the web for the ZX80. There are hundreds of titles for the machine that came after i.e. the ZX81, and thousands for the Sinclair Spectrum, but very little for the ZX80. From this, I settled down with my old magazines, collected for just this purpose and with some scans from Practical Computing magazine from the early 80s, and I played for a while.

Time passed; and then I went back to the web.

And then I found a site talking about a game for the ZX80; a famous game; the famous flicker free space invaders. (See Paul Farrow -Reanimating the ZX80 - this issue)

The site had the code posted as a .GIF graphic. I downloaded it and printed it out. Although it was relatively small, what with it being all in HEX, it took two or three evenings to get it entered correctly and working.

And it was impressive, really impressive.

I did what many do under the circumstances (ahem!), and I made a video for YouTube. Others also thought it was impressive and said so in the comments section. Then somebody spammed the comment section hawking their website (fruitcake.plus.com), and from this I met the internet stranger Paul Farrow.

Not only did Paul have a very deep understanding of the issues involved in making the Space Invaders game, he was also a programmer and he took this knowledge and wrote a game, a flicker free version of PacMan. Not only that he also has a version of Donkey Kong in development too! For the ZX80!

Well blow me down, a modern day developer for the ZX80 and of course he gives all his code away for free. Being lazy and a clumsy typist however, I asked him to put it on a cassette tape for me. He agreed, but then wanted payment from me. Sceptical of this money thing paired with an internet stranger, I hesitantly agreed.

Now I should point out that I think it important that we all support each others efforts in this retro scene of ours, so the very fact that he had written a game and with considerable effort too, was enough for me. But really? Nearly £6 for a copy of a game? Hmm!

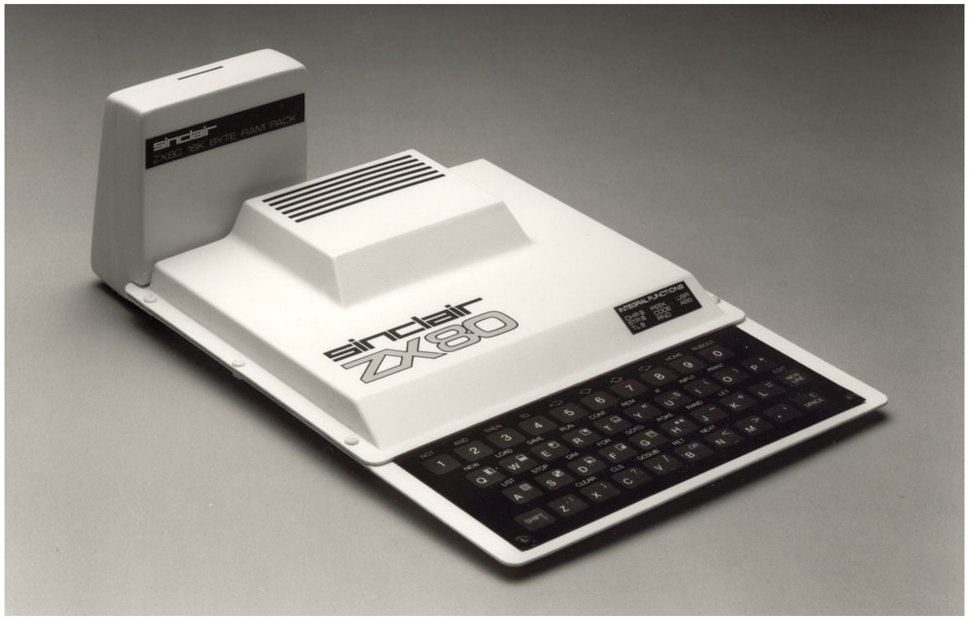
A week or so later the game arrived. I honestly thought I was going to get an old C90 tape that had Frank Sinatra on one side and on the other side, his name scored out and the word PacMan written over it in pencil. Instead I got a professionally crafted package of software, with game instructions on the inlay, cover design, labels on the cassette itself – both sides – and it loaded after only a few efforts (which for the ZX80 is saying something).

And! It is much, much better than the Space Invaders game.

After a week or so he contacted me again to say that there was a bug fix version and that if I sent my purchased tape back he would record the new version for me. I do believe that this is what they call service!

To finish up let me say that one thing, one thing that all of us can probably agree on, regardless of the preferred platform, regardless of whether you are dealing with a gnarly old assembly wizard, or a fresh faced BASIC newbie, this retro scene has some really nice people in it.

Far far nicer than you are ever likely to meet in the Ultimate Cage Fighting scene, that’s for sure.



REAL MEN HAVE FLOPPIES

A BRIEF HISTORY OF THE DISKETTE

by Jostein W. Thorbjørnsen
(with little bits from Dale Goodfellow)

Should you ever find yourself playing trivial pursuit, or having a bar fight argument over just who inverted the floppy drive, one name is sure to come up, that being Alan Shugart, the man commonly seen as the "father" of the floppy disk. While he was the manager in charge of IBM's Storage Development Center in San Jose, California, he himself would be the first to credit David Noble as the actual person who built the first diskette drive, created by using parts from a record player and a tape drive. While it may have been the first it was not however the design that would eventually become dominate.

By the time IBM released their newly developed read-only 23FD "Minnow" drive as part of the System/370 mainframe in 1971, Shugart and most of his engineers had already moved on to Memorex, where they developed the first diskette drive capable of writing to the disk, as well as reading from it. Known as the Model 650, Memorex's drive soon became popular as a replacement for the paper punch cards that were still in common use - a single 8-inch diskette of the time held as much information as a stack of 2000 punch cards, and was considerably easier to handle.

Shugart, seeing the potential for developing this technology further, left Memorex in 1973, in order to start his own company: Shugart Associates. Again, most of his engineers followed him, and Shugart Associates was well placed to quickly take a leading position in the burgeoning market. Shugart's model SA800 8" drive would eventually define the form factor and electrical standard that would dominate the market for decades to come. 1973 also saw IBM release its first read/write diskette drive, but not being involved in peripheral sales for other computer systems, it left that market open for others, a market happily filled by Shugart Associates and in time, many others.

While the floppy disk was originally a product of the mainframe and mini business it was about to gain popularity in a new market, that of personal computing. The early seventies saw the development of "microcomputers" that were affordable enough for computer enthusiasts to own and use at home. At first, paper tape and audio cassettes were the only affordable "mass storage" mediums but were slow and unreliable - (See 300Baud Issue One). When diskette drives became available, they opened up a whole new world of fast and dependable mass storage. Although early drives were often more expensive than the microcomputers they hooked up to, they were of orders of magnitude more affordable than anything else previously available. And once purchased, had the ability to transform a home computing 'toy' into a serious small business machine.

5.25 Inch Minifloppy

By the mid-seventies, word processors were also becoming popular. Essentially these were single-purpose, single-user microcomputers and were designed to sit on an office desk as a replacement for the ubiquitous typewriter.
Dr. An Wang, of Wang laboratories wanted a piece of this developing market, but thought the existing 8-inch diskettes were too large and bulky to fit comfortably in a desktop system. He was also worried that the AC drive motors would cause interference when mounted close to a CRT screen. In response to Dr. Wang's request, Shugart Associates designed the 5.25-inch "minifloppy" as a cut-down version of the 8-inch diskette. There was no new technology, apart from the use of a DC drive motor, so development only took about three months. Interestingly, there are several stories that purport to explain how the size of the minifloppy ended up as five and a quarter inches. Some say it was the smallest size it could be without being able to fit in a pocket - therefore preventing physical damage; others say it was inspired by the size of a cocktail napkin found at a bar during a late-night meeting between Shugart Associates and Wang Labs engineers. A more prosaic, but perhaps more plausible reason, is that the disk drive was designed to fit in the same space as a common tape drive that was 5.75 inches wide.

Whatever the reasoning behind the size, the 5.25-inch minifloppy would soon replace the 8-inch diskette.

In 1977, a young hippie wearing ripped jeans and a dirty T-shirt showed up at Shugart Associates' headquarters. He introduced himself as Steve Jobs of Apple Computer, and he demanded that Shugart Associates (SA) sell him a floppy drive for the ridiculous price of $100. He wanted just the bare drive mechanism, with no electronics, and claimed he had engineers who could design a drive controller with just seven chips. SA agreed, although later Apple switched to Alps Electric Co of Japan, who could supply drives cheaper.

Prior to this, the Apple II had been introduced alongside Commodores PET 2001 at the 1977 West Coast Computer Faire. These first true home computers, were both based around MOS Technologies' 6502 microprocessor, and had similar capabilities. At the time of the

Faire, neither company had a diskette drive, and both computers used audio cassettes as the primary means of mass storage. The PET even had a cassette recorder built into the case, next to the keyboard. The need for a faster, more reliable and convenient means of mass storage soon became apparent, and both Apple and Commodore started work on diskette drives.

At Apple, the job was given to Stephen Wozniak, the designer of the Apple I and II computers. Wozniak had never used a diskette drive, and knew little about how they worked, but together with Randy Wigginton, he came up with a new approach. By letting the computer's microprocessor and operating system do most of the heavy lifting, they could both simplify design and cut costs. The Disk II drive and interface card was launched at the January 1978 CES, at a pre-order price of just $495. Even at the regular price of $595, the Disk II was a bargain compared to other disk drives on the market, contributing significantly to the Apple II's reputation as a "serious" home computer.

Meanwhile, in Pennsylvania, Commodore was developing a diskette drive for the PET. The job was assigned to Chuck Peddle, designer of both the 6502 microprocessor and the PET itself. Peddle took a more traditional approach, designing a dual-drive unit that had its own microprocessor, RAM and ROM, effectively acting as a separate computer that connected to the PET. The Disk Operating System resided in the disk drive unit, and could be controlled from the PET using BASIC commands. Peddle knew that you needed two drives to be able to do useful work with the computer, but later admitted that he should have taken a simpler approach. However in todays retrocomputing world, non-working PET duel floppy drives can provide a valuable source of spare parts for the PET due to this over engineering. By the January 1978 CES, Commodore could only demonstrate a crude prototype of their diskette drive.

WARNING
See it Shoot it
SEE IT SHOOT IT
THIS DISK COSTS £14.99
THIS DISK COULD COST £2,000
It's against the law to pirate software. You can be fined £2,000 or go to prison for six months.
THIS CAMPAIGN IS ORGANISED BY ELSPA EUROPEAN LEISURE SOFTWARE PUBLISHERS ASSOCIATION
Any information on piracy should be passed to The Federation Against Software Theft. Telephone 01-240 6756
PIRACY IS THEFT

Over the next few years, the home computer market exploded with a variety of different computer systems, each with its own diskette drive. Though they all used the now-ubiquitous single-sided 5.25" disks, the data on the disks generally wasn't compatible between systems. In many cases, even different models from the same manufacturer were mutually incompatible. But the role of the floppy disk as the most important storage medium of the home computer revolution was obvious.

At some point, enterprising home computer enthusiasts realised that their single-sided diskette drives effectively wasted half of the storage capacity of each diskette. Double-sided drives were available, but the added complexity made the cost prohibitive. Since most of the home computers used "soft-sectored" disks that didn't rely on index holes or other physical characteristics of the disk itself, the only thing preventing a disk from being inserted upside-down was the lack of a write-protect hole: A notch cut into one edge of the disk, which signalled to the diskette drive that the disk could be written to. The hole could be covered with a piece of tape to write-protect the disk, preventing possible damage to its contents. Cutting a second write-protect notch along the other edge of the diskette was quick work with a hole punch or a pair of scissors, and thus the "flippy" disk was born, along with one of the most stubborn controversies of the early days of home computing.

The debate raged for years between thrifty computer enthusiasts, willing to use their disks on both sides, and more careful users who argued that using the diskettes in ways they hadn't been designed for would put their data at risk. One early argument was that single-sided disks presumably only had been manufactured and tested with single-sided use in mind, and that the "other" side may not be safe to use. Eventually, it became clear that single-sided drives weren't all the same: Some used the "top" side of the disk, and some the "bottom". Thus, single-sided disks had to be usable on both sides anyway. Another argument was that flipping the disk over would reverse the direction of rotation, and potentially dislodge dirt and dust from the soft lining inside the diskette envelope, eventually leading to dirty or damaged diskette drives. Though never really disproved, the widespread use of "flippy" disks suggests that this wasn't really a serious problem.

By the early 1980s, the home computer market was thriving, and the market for storage grew along with it. Many companies tried to design new kinds of diskettes, to address one or more of the shortcomings of the 5.25" floppy; vulnerability to physical damage, size, and storage capacity. A number of different designs were proposed, and most of them foundered in the wake of the 800-pound gorilla that was the 5.25" format. The diskette drive and media manufacturers eventually formed an industry association to review the many new formats, and suggest a new standard.

One popular format, particularly in Europe was the 3" disk, this was adopted principally by Amstrad who used them in their CPC and PCW systems and also in the Spectrum 3 home computer. However the popularity of the 3" disk was really only sustained by the popularity of the various Amstrad systems, when they stopped using them their mass market use also disappeared even though other machines such as the Tatung Einstein also used them.
In addition, while the 3" drive clearly had an advantage over the 5.25" disk with regards to size, in other ways it was less attractive, in that it held around the same amount of data yet was more expensive as it was harder to make. It was another "also-ran".

Sony actually introduced the the 3.5" disk in 1982 but this was a slightly different design from the consensus decision eventually settled upon by the drive and media associations. When this decision was made it then had the same problems as every other disk format before, namely the sheer size of the 5.25" installed based. It was used in a few machines, but it was still having the same troubles as other formats before it. It was only when Apple agreed to use it in the newly developed Macintosh and then both Atari and Commodore bought into it with their 16bit machines that the format finally gained the traction to make it the market leader.

And the rest, as they say is history. 3.5" drives continued largely unchallenged for a couple of decades, there were efforts to unseat it with Zip drives from Iomega or Floptical disks from Insite Peripherals which were a like a cross between a floppy disk and a magneto optical disk, or even magneto optical -MO- disks themselves, but ultimately they held their ground until the arrival of the solid state USB memory drive. Today, 3.5" disks can be bought easily online by weight, such is their commonness, 5.25" disks while still around are beginning to command a price on auction sites for a fresh box of ten and 8" disks are an altogether rarer breed, usually sold singularly and untested. This perhaps should tell us that we should probably start hoarding the 5.25" and 3.5" disks available now, while we still have the chance.

GETTING STARTED WITH ASSEMBLY UNDER CP/M

by Jonathan Chapman

So you've got your CP/M system up and running, and you want to do some Assembly programming. Perhaps you have some past experience in 8080, 8085 or Z80 Assembly, or you've at least used the CP/M assembler to build custom versions of software, such as adding an overlay to MODEM7. But how does one go about writing Assembly programs that interact with the interfaces that CP/M provides, rather than talking to the hardware directly?

I'm currently at a point in which I'm starting to write Assembly for CP/M using my Kaypro II. I've got experience in 8080/8085 Assembly, but only in writing code to be executed directly on hardware without an OS. In seeking to learn how to take advantage of CP/M's resources in my programming, I was unable to find a good low-level guide. Even a simple "hello, world" doesn't seem to exist for CP/M Assembly. To try and fill this gap, I'll be writing a series of articles (to be found on my personal site) hopefully to help others get started as I myself learn. This first article will focus on getting a working environment set up, including Assembler options, editor options, and a brief "hello, world" example.

EXPECTED KNOWLEDGE, HARDWARE AND SOFTWARE

These articles assume a working knowledge of 8080, 8085 or Z80 Assembly and a familiarity with good coding practices for the architecture of your choice. In order to reach the largest number of readers, all examples will use 8080 opcodes and Intel mnemonics. Similarly, unless otherwise stated, the assembler and linker included with CP/M-80 2.2 will be used to assemble all examples and link them into .COM executables. Thus, the only hardware required to code along is a machine capable of running CP/M.

Aside from the default tools provided with CP/M-80's 2.2 version distribution, I've found it essential to use an external editor. It's possible to write the short examples included within using ED, but I wouldn't recommend it for larger applications.

Example code will be available for download through The Glitch Works, my personal website. It can be found at http://www.glitchwrks.com/vintage/ – code will be posted along with any freeware/shareware archives discussed in the article.

CHOOSING YOUR EDITOR

Most people who have attempted to do any sort of programming under CP/M quickly find out that the included editor, ED, is overly complicated for all but the smallest of tasks. As such, the first step in setting up a working environment for programming Assembly under CP/M should be the selection of a text editor. In my quest to find an editor that I'm comfortable with, I came across many options, but will comment on three below.

My first text editing experience with the Kaypro was through WordStar. While WordStar can write WYSIWYG word processed documents, it's also capable of producing unformatted text files when set to non-document mode. I found this a bit cumbersome, which is understandable as WordStar wasn't really designed with programmers in mind. It's far better than ED, but I feel there are better options.

I later received two recommendations from members of the Vintage Computer Forums: ZDE and VDE. Both are intended as general-purpose text editors, and work well in the capacity of writing programs. Both support many terminal configurations and WordStar key bindings. I've tried both, but ended up settling with VDE. ZDE is available through Bo Zimmerman's CP/M archive, while VDE is available from Eric Meyer's Google Code page (there are DOS and Windows versions that seem to be fairly current, but the last CP/M version dates from 1988).

My eventual choice to stick with VDE was based on speed: for me, it loads faster, searches a document faster, and generally handles word wrapping better than the other options I tried. It's written in Z80 Assembly, though, so if you have an 8080 or 8085 based system, you won't be able to use it.

Installing your text editor consists of obtaining the installation archive, unpacking it, writing the files out to disk and configuring the editor for your particular system. I did so by uncompressing the LBR file locally on my Linux machine, then sending the files to the Kaypro using Kermit-80. You can use 22DISK and a DOS or CP/M decompression utility as well. One thing to note about VDE is that I was unsuccessful in getting the memory-mapped version to work with my Kaypro II, and had to instead configure the standard version.

THE ASSEMBLER AND LINKER

Fortunately, CP/M includes everything you need here for basic programming. These tutorials and examples will rely on ASM.COM and LOAD.COM included with CP/M-80 version 2.2. These tools, along with your text editor, are all you need for writing 8080 Assembly that will take advantage of CP/M's operating system resources.

You may choose to use Microsoft's M-80 assembler, a Z80 assembler that supports Intel 8080 mnemonics, or adapt the provided examples to your assembler of choice. The CP/M assembler is a little limited in that it's tailored for producing files that will be executed from CP/M, so if you're wanting to burn your assembled program to ROM and run it without CP/M, you'll need something else.

SETTING UP YOUR WORKING ENVIRONMENT

To write programs effectively with vintage hardware, it's necessary to give a little thought to your programming environment. Systems supporting higher- capacity floppies or hard drives don't need to worry about this so much, but most floppy-based CP/M systems will. The space limits imposed by single-sided floppies, especially if you're using 5.25" media, means you'll need separate floppies for your editor, the assembler/linker, and your source files.

In my case, I use three floppies for my programming environment. The first is a working copy of the CP/M 2.2 system disk distributed with the Kaypro II. It provides file system tools (PIP, STAT, DIR, et c.) as well as the CP/M assembler and linker. This disk is write-protected, since you shouldn't need to change any of the data stored on it. The default programs included on the Kaypro distribution media leave a few kilobytes free.

Second, I've got a floppy that contains VDE with the Kaypro-centric overlays installed, the VDE quick reference, PIP, STAT and a program I wrote to disable the Kaypro keyclick sound (does anyone actually like the dying cricket noise used for keyclick?!). Since VDE is fairly small (less than 20 KB) there's still room left on the working copy for text files, but I have the floppy write-protected anyway, since I don't use it for this purpose.

The final disk in my setup is the working disk that contains my source code, assembler listings and assembled code. While the other two floppies are bootable and contain various executable utilities, my working disk contains only code and assembled/linked programs. Most of the time, this disk will remain in the B drive through reboots.

Another useful option, though not essential to programming, is including a terminal program on either your editor floppy or a fourth disk. I keep Kermit-80 as well as a dumb terminal program on a fourth bootable floppy for my Kaypro. This allows me to transfer source code and assembled programs between the Kaypro and my Linux desktop system.

Essentially, with your working environment spread across several floppies, you'll be leaving your working disk in drive B while switching the disk in drive A between your assembler/linker disk, your editor disk, and possibly a terminal transfer disk. Remember, each time you change disks, you need to do a warm reboot with Control + C so that CP/M may log the changed disk...if not, you risk ending up in a situation where CP/M can't write data to a floppy (you'll get a BDOS error), potentially causing you to lose work!

Those with DSDD 8" drives, quad density 5.25" drives, 3.5" drives or hard disks probably won't have to worry about swapping floppies as much. You may be able to get away with a single floppy for assembler/linker, editor and source code storage! There's nothing wrong with that, but do remember to back up your data, since there won't be as much separation between environment and user code/programs.

THE BDOS: HOW ASSEMBLY PROGRAMS WORK WITH CP/M

Now that we're ready to write and assemble programs, it's time to take a look at how Assembly programs actually interact with CP/M. Since CP/M is a single- tasking operating system, there are no calls to daemons or services running concurrently in memory. Instead, a portion of the CP/M operating system, the BDOS, is left resident in memory. Its location in memory is universal across various implementations of CP/M of the same version, so assembled code for one machine will still execute on another, as long as it doesn't make hardware specific calls.

We can thus consider the BDOS as a layer of abstraction above the physical hardware. Calling a print string function in the BDOS invokes a subroutine that handles sending characters to a machine's display...while this can be very machine-specific on the hardware level, using the BDOS provides a generic way to do so. Instead of interacting directly with the display controller or console serial port, we can call the BDOS to print a string we provide and let it take care of the details.

Assembly programs may call the BDOS by placing a function code in the C register, a parameter in the DE 16-bit register pair, and calling 0x0005. This address is the BDOS entry point for calling its subroutines. In practice, it's usually best to assign the BDOS entry point with an equate. This makes it easier on the programmer as well as anyone reading your source code in the future. We do so as such:

```
BDOS      equ  0005H
```

When the BDOS is called, the function it performs is determined by the function number in the C register. CP/M 2.2 provides 39 separate functions, the description of which may be found in various CP/M manuals and online databases. Variables are passed to the BDOS using the 16-bit DE register pair. For functions that return only a single byte, the return value is placed in the A register. Functions returning a 16-bit value do so using the HL register pair.

While not directly expressed, an Assembly program called from the CP/M command line will return to the CCP command processor through a RET, due to how the CCP maintains its software stack. Programs can also return to the CCP by jumping to the BOOT address or 0x0000.

PRINTING A STRING: A SIMPLE BDOS EXERCISE

We're now equipped with all of the information we need to begin utilizing the CP/M BDOS functions in our Assembly programs. Since we're going to use the underlying CP/M system, we need to make sure our program does not load in such a way that it overwrites the resident CP/M code. To do so, we must tell the assembler that our program is to start at the beginning of the Transient Program Area, or TPA. The TPA begins at 0x0100, and we specify the starting address as such:

```
ORG  0100H
```

Our example program will declare the BDOS entry point, define a string terminated by a dollar sign ($), load the integer that represents the BDOS string printing function (9) into the C register, load the address of our string into the DE register pair, and call the BDOS. Once it's done, we can RET to give control back to the CCP.

```
BDOS      equ  0005H
ORG       0100H
START:    mvi  C, 9
          lxi  d, msg$
          call BDOS
          ret

msg$:     db   'hello, world$'
end       START
```

That's it! We can now assemble our program using the CP/M assembler:

```
A>asm B:hello
CP/M ASSEMBLER - VER 2.0
0117
000H USE FACTOR
END OF ASSEMBLY
```

This gives us two new files: HELLO.PRN and HELLO.HEX. HELLO.PRN is an assembled program listing of our demo, providing the address of each operation. HELLO.HEX is the machine code output file that contains our program in 8080 machine language, represented in the hexadecimal system. We can then link the machine code file using LOAD.COM to produce a CP/M executable:

```
A>load B:hello

FIRST ADDRESS 0100
LAST ADDRESS  0116
BYTES READ    0017
RECORDS WRITTEN 01
```

The disk in the B drive now contains HELLO.COM, an executable file. By running this file, we'll call the BDOS to print our string to the CP/M console:

```
A>B:hello
hello, world
```

Since our program is small, there's no need for a warm boot, and we can return directly to the CCP.

There you have it – a short introduction to the functionality of the CP/M BDOS and an example of how to call its functions! For a list of the references used in this article, future articles, and general vintage computer fun, please visit *www.glitchwrks.com/vintage/* and follow the links for this series!

300 BAUD

OUR BBS TOWN

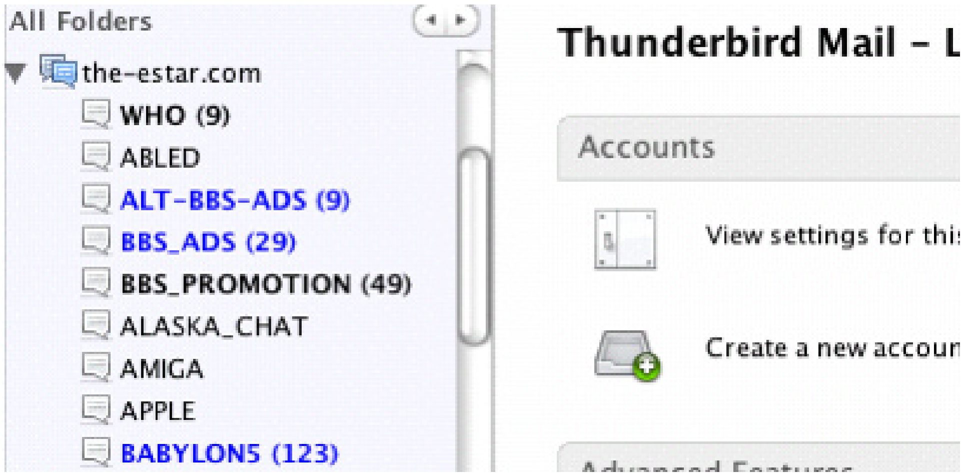
By Greg Goodwin

Virtual towns made by users for users. This is the ongoing account of how the homespun computer network, the BBS, made available to users before the internet went public still lives on today. Please tell us about your experiences with the BBS's in current day by writing to greg.goodwin@rdfig.net, which is ironically an E-mail account on a BBS.

This time around we will be looking at two subjects on the Computer Bulletin Board System: 1) How to receive FidoNet, DoveNet, and other BBS message networks through a newsreader like Mozilla's Thunderbird, and 2) The debate over the plural of 'BBS'

Part 1 - Bringing The BBS Experience To You

One of the really great things I've learned recently about the BBS scene is that I could receive BBS networked message traffic as easily as I could receive an E-mail message or a newsgroup message. Naturally I love to call the BBS's, and get the full experience, however, sometimes my time is short at home between my video games, TV watching, and visiting friends that my only time to read my BBS traffic is at the less hectic pace of work.



When the following advert was posted on a BBS Promotion board and then by following the instruction on the website, I've been able to have wonderful discussions on the recent Doctor Who episodes (along with other subjects and message areas) and at the same time keep people company out there on the "BBS"s. I simply load all the messages before I hit the road, read them and write responses while when I'm out and about and have them sent sometime later when I return home.

If you are not a cheapie like me and have bought some sort of wireless connection on the go, you can receive and send on the road practically instantaneously.

If you need any help in setting up your newsreader to these instructions, then I received some good help on the DoveNet boards, but you can write us here at Our BBS Town at Greg.Goodwin@rdfig.net.

```
POSTED 8/22/2010 10:59 PM
Fidonet In Your Newsreader
==========================

Outlook Express
Forte (Free)Agent
S & H News Rover
Mozilla Thunderbird
Others

Read and Post to Fidonet Message Echoes From Your Favorite NEWSREADER.

* From Anywhere

* No Massive Message Control Lines

* Messages Are 100% Pure Fidonet, Are NOT Gated To Or From Internet
Newsgroups Or Other Networks.

To Sign Up Goto:

http://www.easternstar.info

You Can Check Out The System Anonymously (READ ONLY) By Pointing Your
Newsreader To:

the-estar.com On Port 1119.

================================
```

Part 2 - The Plural Of 'BBS'

For all you retro computer fans out there, many of which can remember back to the glory days of BBS's I am happy to report that one aspect still continues in a way that you will remember quite well and that is the knack to have the occasional strong opinionated colourful person in the ranks.

Now I promised our patient publisher Dale Goodfellow that I would not wax nostalgic on the glory days of the BBS, and naturally my article is to encourage BBS activity in the here and now. But it is just fun to throw in a fact that was as true in 1982 as it is now; that you have your colourful characters out there, both in the good ways and in the bad, and that this is really what has made and continues to make BBSing interesting.

One of these colourful characters has a 'BBS promotion' site on Facebook where you can post information about a current BBS, and then have it criticised negatively and raked across the coals by the person running the site.

Mind you, this person shows only one bit of the colour of the current BBS users, but this broken crayon gets us into the story and where we need to be.

So I advertise a BBS and how to get there, the broken crayon posts back a comment "Can't do anything once you get there". Now this broken crayon of a colourful individual is known for ripping down what he does not immediately approve of, but ironically in this one case the BBS literally was not responsive once you got a connection. However he had such a history of being negative that I said, "Your negative comments do not help promote BBS's"

In my E-mail box I saw that I got three separate responses to my comment, the last comment said "and it's BBSES, not BBS's"

Naturally this had me curious, so I started to write back only to find I was not only off the site, I was blocked so that I could not find the site.

There is a old saying:
"If I speak in tongues of men and of angels, but have not love, I am only a resounding gong or a clanging cymbal." (Bible / I Corinthians, chapter 13)

This broken crayon might have been right, but with no way to ask him, his message almost came off with no effect on me. In all my years of BBSing since 1984 I had never heard of that. I've been calling BBS's for 26 years and I hope to call them at least 20 more so it seemed a good time to find the correct term from this point forward.

First off, a quick search on the Internet showed that many people call the plural of the BBS, "BBS's". If you, like me, have been doing this, you are not alone, and this it seems is common to do. For almost tradition sake I will probably refer to the days before now as the BBS's and say it with a smile.

Looking at the web page below it states that " 's is a form of possessive when attached. "

http://www.wikihow.com/Use-Apostrophes.

Also:

"Never use an apostrophe to indicate a plural. The incorrect use of an apostrophe to form the plural is called the greengrocer's apostrophe..."

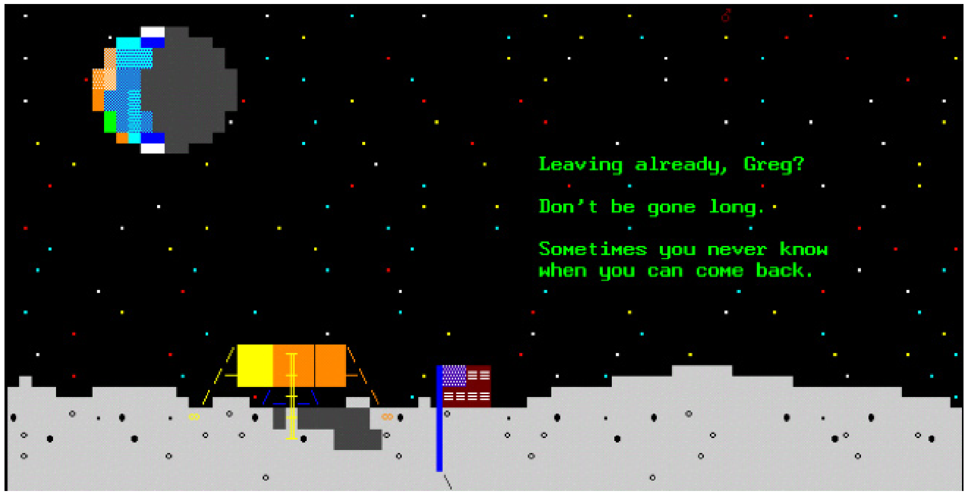
So from that, BBS's would in fact be incorrect.ˇ

"An exception to this use is in the case of making a single letter plural. Therefore, "Why are there so many i's in the word "indivisibility?" is correct.

This is simply for clarity reasons, so the reader does not mistake it for the word "is." However, in modern usage, the preference is to avoid inserting an apostrophe and instead surround the single letter in quotation marks before pluralizing it: Why are there so many "I"s in the word "indivisibility"?

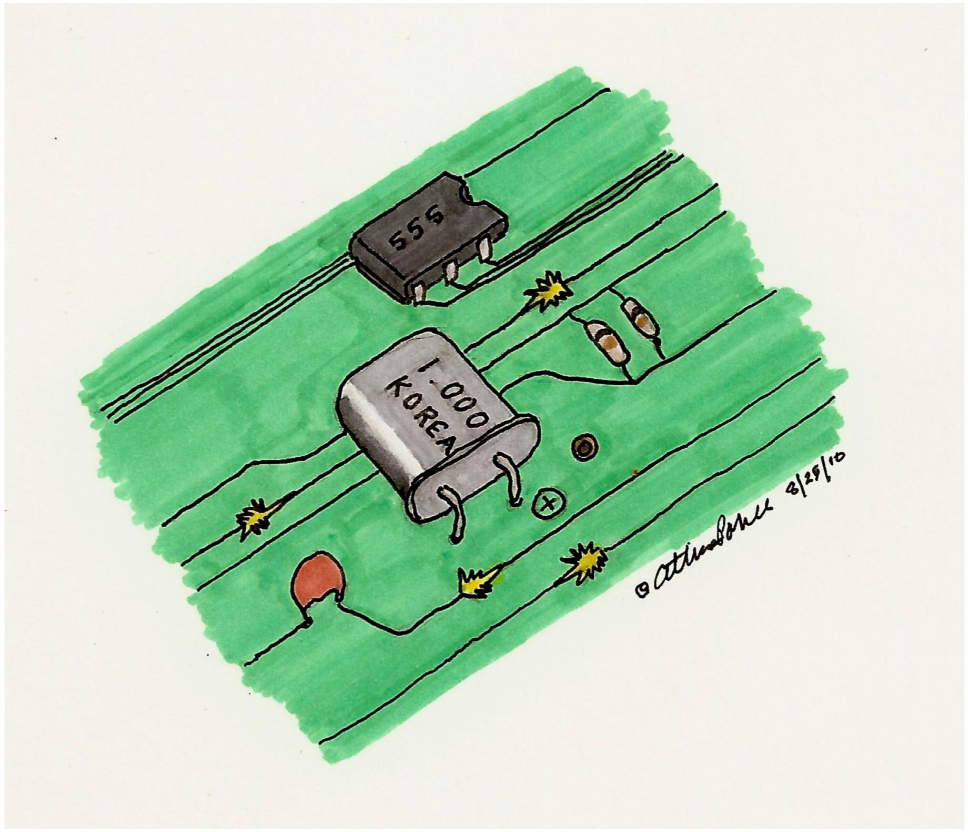
Similarly, apostrophes can be used when talking about a word (e.g., This list contains a lot of do's and don't's) but quotation marks can make it clearer ("do"s and "don't"s)."

So with that, "BBS"s would actually be more accurate than what has been previously suggested with BBS's or BBSes. Good news however is that all three entered into a search engine will help you find some good BBS material between now and the next article. Until then, I hope to see you out there on the BBS's; in those virtual towns made by users for users.


Leaving already, Greg?

Don't be gone long.

Sometimes you never know
when you can come back.



300 Baud Magazine
1 Yester Place, Haddington
East Lothian, Scotland, UK, EH41 3BE
www.300baudmagazine.com

Editor William Dale Goodfellow • **Illustration** Atticus Pamer • **Layout** Simon Williams
Production by LD8 Media Design • www.ld8.ca • Printed in Canada

History of Commodore Computers Poster 24" x 36"

The Commodore systems have been professionally photographed specifically for this poster and the photos appear nowhere else. This poster is ideal for any vintage computer fan, Commodore Computer collector, classic or vintage memorabilia display. This is a limited edition poster and will not be re-printed! Poster includes hi-resolution images of rare and hard-to-find systems and selected diskette drives.

Order Online: http://www.vintagecomputer.net/poster_detail.cfm $19.99 US plus shipping (US, Canada, Worldwide flat rate)