UNCLE BERNIE'S APPLE-1 TOOLCHAIN --- PART 1

MOTIVATION

In the past years, several Apple-1 owners have asked me about the tools I use to develop my "exploits" for the Apple-1. So there seems to be a demand for an Apple-1 toolchain allowing quick development and test of Apple-1 machine code programs.

Some months ago, Bobby Nijssen contacted me about his project [1] of a keyboard emulator cable [2] which plugs into the Apple-1 and can download Apple-1 software directly from the Apple-1 software library "Apple1Software.com" [3] he has created and populated with public domain software. He asked for permission to use my "Autotype" and "TurboType" systems which I had developed for my own keyboard emulator cable [5] needing a host machine with a parallel printer port, and Microsoft DOS (or a free DOS). Bobby's development does not need such long obsolete hardware and software, as it connects to a modern computer which has Internet access - its manual can be found in [4].

To support his project, and to enable Apple-1 enthusiasts from around the world to write and contribute their own Apple-1 software to be published on Bobby's website (or elsewhere, such as on the Apple-1 forum of www.applefritter.com), I brought some of my own software development tools up to snuff, which means, made them ready to be used by other software developers, and I also wrote this manual for this toolchain.

This first part of Uncle Bernie's Apple-1 toolchain runs under Linux and automatically cross assembles a 6502 assembly language source into APL and TUR files which can be auto-typed into the Apple-1 via a suitable keyboard emulator cable. These files can also be cut and pasted (or at least loaded) into most Apple-1 emulators available on the web, and – if you choose to do so - be published on Bobby's "Apple1Software.com" web page [3].

Have fun using these tools to create more nice Apple-1 games and tools !

Colorado Springs, January 2025
'Uncle Bernie'

REQUIREMENTS AND LIMITATIONS

This toolchain runs under LINUX and all tools are compiled from the sources. Since I can't test this release on all known Linux versions, you may need some Linux know-how to make it work on your version of Linux. I use a 64 bit version of Linux Mint (19.3). But I have kept everything as simple as possible and don't use any gimmicks or exotic tools. I also tried to keep the shell scripts as linear and simple as possible – no loop, no function calls, no difficult constructs. You need 'make', 'gcc', 'patch', and the libraries libz-dev and libsdl-dev available for any newer Linux version. Note that some of the tools in the toolchain were built by third parties (even I refuse to re-invent the wheel). These are 'open source' and must be downloaded from the internet, so you also need some form of internet access. After the downloads, no internet access is needed anymore to build the toolchain.

What you can do with this "Part #1" of Uncle Bernie's toolchain is:

- use your preferred text editor to write assembly language source code for the Apple-1,

- run a script which invokes an assembler and makes APL and TUR files from the binary,

- download the APL or TUR file into an Apple-1 emulator or a real Apple-1.

For download into a real Apple-1, you need one of the various keyboard emulator cables.

What you can't do yet with this toolchain is to generate a WAF or AIFF audio file for download of your program into the Apple-1 via the ACI. Be patient, this functionality will become available soon, or, if you have an older, suitable 68000 based Mac, you can try to compile [6], but be aware, this program written by Mike Willegal many years ago won't work on non-68000 CPU architectures. And please don't ask Mike to fix it – I already got his permission to modify and publish a 'endianess' independent version of his 'toaiff.c', but got stuck with an Apple II issue it has that needs to be resolved before I can release it.

HOW TO GET AND INSTALL

You need (as a minimum) to download the following source packages:

1. Mark Schmelzenbach's 6502 cross assembler, named 'atasm' [7].

2. The POM-1 Apple-1 emulator written by John D. Corrado and Verhille Arnaud [8].
   (its use is optional, but recommended for testing your code creations – I use it, too.)

3. The 'UB_toolchain_part1.zip' file [9].

These can be obtained from:

https://github.com/linuxha/atasm
(... by clicking on 'code' and then download the file: atasm-main.zip)

https://sourceforge.net/projects/pom1/files/pom1/1.0.0/pom1-1.0.0.tar.gz/download
(… which should give you the file: pom1-1.0.0.tar.gz)

https://www.applefritter.com/content/uncle-bernies-apple-1-toolchain-part-1
(… which should offer you this pdf and the file: UBTOOLCHAIN_PART1.ZIP)

Once you have obtained these files, put them into a newly made, empty directory on your Linux machine. You may use any name for that directory. Let's assume it's 'BUILD' (… you may note I have a method with uppercase and lowercase file and directory names). Type the following commands, but modify the 'cp' destination path names as needed by your choice, or use a mouse pointer based desktop tool to copy the three downloaded files into your 'BUILD' directory (hint: be lazy and use TAB key to complete file names):

```
mkdir BUILD
cp atasm-main.zip BUILD
cp pom1-1.0.0.tar.gz BUILD
cp UBTOOLCHAIN_PART1.ZIP BUILD
cd BUILD
ll
```

You should have some output like this:

```
drwxrwxr-x 2 john doe   4096 Aug 17 18:22 ./
drwxrwxr-x 8 john doe   4096 Aug 17 18:07 ../
-rw-rw-r-- 1 john doe 332202 Jul 27 12:54 atasm-main.zip
-rw-r--r-- 1 john doe 111925 Aug 10 18:38 pom1-1.0.0.tar.gz
-rw-r--r-- 1 john doe  24230 Aug 17 16:29 UBTOOLCHAIN_PART1.ZIP
```

The user name and the dates and the dates will change, of course. But the file names and the file sizes should be the same. Observe the uppercase and lowercase. Case of the first two filenames is important and cannot change (the scripts require that to match). If they differ from the above, rename the files to the right case (that is: lowercase, as seen above).

There always is a concern that malevolent hackers inject malware into any published software. I don't see a risk for any published Apple-1 software, as when you turn the Apple-1 off, any worm or virus will be gone. But for Linux work, we need to be a bit more prudent. This is why I provided a SHA-2 checksum for my toolchain download:

To verify that what you got is what I wrote, and not some file that hackers have tampered with, give the following command – a copy paste from this document is recommended to avoid typing errors (I've used a very small font to get it into one line so copy/paste should work under all conditions - don't try to type in this tapeworm manually):

```
echo "a8505f640ee12f4136dff1cebb38df342ae74c7c668b2fac798d1aa39b1f77bf UBTOOLCHAIN_PART1.ZIP" | sha256sum -c
```

The output should be:

```
UBTOOLCHAIN_PART1.ZIP: OK
```

This tells you with high confidence that nobody has tampered with that zip file. Any other outcome means it's not the same file I put up for download. Don't use that file any further – it may contain malware - and alert us by "send PM" on www.applefritter.com to 'Tom Owad' and 'UncleBernie', and mention the failed SHA-2 check. But please, do this only if you are 100% sure that you did copy the above command line properly – the smallest mistake in the long hexadecimal number will cause a false alert ! I have found that if a larger font is chosen, and the command needs a 2nd line, some pdf readers will insert extraneous whitespace or other control characters in copy/paste process, which makes the SHA-2 check fail. This is why I made the font so small that the above command fits into one line. Be aware of this bug / pitfall !

If you got an OK response as seen above, give the following command:

```
unzip UBTOOLCHAIN_PART1.ZIP
```

Now, use your favorite text editor to edit the 'UBSCRIPT' which came out of the zip file to put the tool chain in a new directory which suits your needs. The relevant line is at the beginning of the script:

TARGET=$HOME/ubtools

… which you can change at will, as long as the TARGET will be a valid path for this user, and the 'ubtools' would be the name of the new directory in which the script will create to put in the tools. In other words, you can replace the string 'ubtools' with your own target.

The script will check the authenticity of the third party downloads with SHA-2 and then automatically build the tools (and POM-1 if its tarball file is there) and move them (and support files) to your TARGET directory. If anything breaks and errors appear, bad luck.

In this case of bad luck, try to find out what is amiss and then run the build manually. The necessary commands are all in the 'UBSCRIPT', you may just copy/paste them line by line to find out where the problem is. This is why I kept the script simpler than I could, and it got longer than necessary by massive use of copy/paste for the same type of function blocks. This is inefficient coding but more user friendly, especially for Linux newbies.

If you happen to be such a Linux newbie: any missing libraries, packages, etc. can be added to your Linux installation with the command 'apt-get': consult online tutorials how to do this, it's very easy, but requires system administrator / root credentials.

If you already have POM-1 installed, please delete its tarball now, to avoid trouble with double installations I did see when doing my trial runs:

`rm pom1-1.0.0.tar.gz` (DON'T DO THIS UNLESS YOU HAVE POM-1 INSTALLED !)

Then run the script using the commands:

```
chmod +x UBSCRIPT
./UBSCRIPT
```

This will automatically build the tools and move them (and their support and documentation files) to the TARGET directory. If anything breaks and errors appear, bad luck. Try to find out what is amiss and then run the build manually. The necessary commands are all in the 'UBSCRIPT', you may just copy/paste them to try step by step. Please don't ask me for help with that – I simply can't help as I don't have your machine.

After building the tools, the script will also run a test case to automatically verify that the tool chain works as expected. This hinges on the 'TRIAL.APL' and 'TRIAL.TUR' files from the zip file. You may delete them once everything works.

If you have provided the file 'pom1-1.0.0.tar.gz', the script will also unpack the POM-1 emulator source code for you, run its makefile, and install it, so it is ready to use. It will try to start POM-1 from the 'UBSCRIPT', and if this succeeds, then a POM-1 window will pop up, and POM-1 is ready to use. Keep it for your first attempt to program the Apple-1.

(If no window pops up, something failed with the POM-1 installation. If so, check that the POM-1 executable has been generated – it should be in the pom1-1.0.0/src directory and be named 'pom1-1.0.0', and it should be executable. If you find it there, then the compile worked, but the 'make install' failed because it did not have the required authority. You may try 'sudo make install' from that directory – but do this at your own risk, as this command gives superuser authority to the install script which came with POM-1.)

Once POM-1 is installed, you can invoke it at any time with the command:

```
pom1
```

… and the POM-1 window will appear. If you click on the window, you can type commands as you would do with a real Apple-1. Try, for instance, the small test program found in the Apple-1 manual. CTRL-R will reset the emulated Apple-1, just in case.

If POM-1 does not work on your system, sorry, I can't help you, so don't ask me for help. See, it wasn't me who wrote POM-1, and I am not going to spend any of my time fixing it for you. For me, it did work out-of-the-box, on all Linux machines I tried it on. It does

have some quirks, but these will be explained later, so you don't fall into these traps as I did, when I tried to figure it out what works and what doesn't. Still, it's a nice emulator.
If you don't like POM-1, or if you can't make it work, there are other Apple-1 emulators out there. Try them. Some are for plain online use, no installation required, like [12]: this one has a copy/paste feature invoked with the "LOAD" button, and you can copy/paste whole APL and TUR files into the white text window that pops up.

HOW TO USE THE TOOLCHAIN

This involves the usual software development cycle:

Step 1: Write or modify your source code using a text editor.
Step 2: Run the assembler, and if it complains, go back to Step 1.
Step 3: Load your program into the Apple-1 and run it. If it has bugs, goto Step 1.
Step 4: Prepare WAV and AIFF files for release of the software.

(Part 1 of Uncle Bernie's Apple-1 toolchain does not support Step 4 yet, as my own tool for it, 'ACIace', is not ready for release, and the modified-for-Linux version of Mike Willegal's 'toaiff.c' [6] is not yet ready either, as it makes AIFF files which fail to load on an Apple II, but load fine on an Apple-1, and no, it's not the checksum for the Apple II).

Let's explore this development cycle (assuming we are in your TARGET directory). First, we will take the code skeleton and make a derivative. Here is how it is done:

go to your TARGET directory with the toolchain and give the commands:

```
cp a1_skel.m65 mytype.m65       (make a copy of the skeleton)
chmod +w mytype.m65             (make it writable so you can edit it)
vi mytype.m65                   (invoke text editor)
```

You may use any other plain text editor available under Linux, but you can't use those who use any file format other than plain ASCII text. So you can't use 'OpenOffice' and the like. Real programmers don't use such fancy WYSIWYG editors to write code, instead, they use real text editors like ed, vi, emacs, etc. to write code. (Uncle Bernie prefers 'vi').

Using your text editor, find the string 'TYPEWRITER' which is the assembly language line:

```
        .BYTE "TV TYPEWRITER", NL
```

... change it to your name:

```
        .BYTE "JOHN DOE'S TYPEWRITER", NL
```

... and then save the source and exit the text editor. We are done with Step 1.
Now Step 2:

Give the command:

```
./asm65 mytype
```

... to invoke Uncle Bernie's Apple-1 toolchain. 'asm65' is a shell script which first runs 'atasm' to produce a binary file (here: ‘mytype.bin') containing the object code for the 'mytype' program we just wrote, and then invokes a few common UNIX utilities like 'sed' and 'grep' to extract information from the various output files of 'atasm'. These are fed into Uncle Bernie's 'bin2wom' tool which makes the output files 'mytype.apl' and 'mytype.tur', but only if everything went fine. The shell script does not attempt to catch errors. This is done to keep it simple. So if 'atasm' pukes on your source code, or if your source code does not have the BEGIN and START labels at the correct places, the outcome may be weird, and stale output files from a previous run may be there. So watch out for error messages. No need to delete stale output files … they will go away once a run is error free.

If everything is right, then you should get the following screen output:

```
ATasm 1.08 (A mostly Mac65 compatible 6502 cross-assembler)
Pass 1: Success. (0 warnings)
Pass 2:

Assembly successful
  Compiled 115 bytes (~0k)
  Writing raw binary image:
    0800-0872

Compiled to binary file 'mytype.bin'

Uncle Bernie's 'bin2wom' tool invoked with the command line:
./bin2wom mytype.bin -i
./bin2wom generated output file: 'mytype.apl'

Uncle Bernie's 'bin2wom' tool invoked with the command line:
./bin2wom mytype.bin -i -t
./bin2wom generated output file: 'mytype.tur'
```

Now it's time to load the new program into the Apple-1. For the real Apple-1 hardware, move the 'mytype.apl' and 'mytype.tur' files to the host system of your keyboard emulator cable, and follow its instructions. For Uncle Bernie's keyboard emulator cable driver [5], use function keys F1 and F2 to clear the Apple-1 screen memory and to reset it, which starts the 'WOZmon' residing in its PROMs at location A1, A2 (lower left corner of the Apple-1 motherboard). Then press F3 to invoke the download process, and give the appropriate file name. The 'mytype.apl' is plain WOZmon command line syntax and loads slowly. The 'mytype.tur' uses Uncle Bernie's "TurboType(tm)" system which loads user

programs much, much faster, but there is overhead for the loader and the crc checker, so the benefit of faster loads only appears with longer programs. This is the reason why both types of files (APL and TUR) are supported. For small programs, use the APL file.

To get the POM-1 emulator, you may need to start POM-1 by giving the command:

```
pom1
```

… unless it is already running. POM-1 is a window with green text and a blinking '@' cursor, like on a real Apple-1 using a green phosphor CRT monitor (for the readers tender of age, a 'CRT' = “Cathode Ray Tube” was a large, heavy and dangerous vacuum tube to display pictures in TVs and computer monitors. It also doused the viewer with a small dose of soft X-Rays. After Y2000, CRTs were replaced by LCD based flat screen displays).

Then, click on the POM-1 window, and type CTRL-L. POM-1 will respond with:

```
Enter file to load:
```

Type the file name (either 'mytype.apl' or 'mytype.tur') and press <return>.
POM-1 will respond with:

```
Choose file format:
Press 1 for ASCII or 2 for Binary
```

So, press key '1'. POM-1 will respond with:

```
Do you want to simulate keyboard input?:
Press 1 for yes or 2 for no
```

If you press '1', which works as expected, and is preferred, then POM-1 will emulate the real Apple-1 with its slow screen output rate (fast enough for human typists, though), and you can watch everything as it would happen on a real Apple-1 running the WOZmon. This of course implies that POM-1 is indeed running the WOZmon, and is not in a crashed state or running another program. If POM-1 does not show auto typed characters on the screen, reset the emulated Apple-1 by CTRL-R and repeat the above sequence.

If you press '2', then POM-1 will not emulate the slow screen output, and is supposed to load the program faster. Alas, choice '2' does not work as I would expect. It seems to be able to properly load an APL file – don’t try that with TUR files, it won’t work - and so POM-1 seems to understand to parse for addresses and data bytes in the file, but it does not accept the WOZmon 'run' command. Which you still can give manually. If you have loaded 'mytype.apl' using option '2', just give it the required run command:

```
830R <return>
```

... and you will get the following screen output:

```
0830: 78

JOHN DOE'S TYPEWRITER
ESC FOR EXIT
@
```

... where '@' is the blinking cursor. You can now type any text you want, and as long as the characters are supported by the Apple-1, they will appear on the emulated screen, as you would expect from a "TV Typewriter", which was a fad of the early 1970s, with limited usefulness, except if hooked up to a real computer – which almost nobody had at the time being. But for people of the early 1970s, it was absolutely exciting and fascinating to be able to type on a keyboard and see the text appear on their own TV set. It was like magic !

I remember, I was there, and was blown away, so I did choose this as an example program.

Note that the Apple-1 has no cursor movement capability other than one step forward (after accepting a new character) and to the next line down. Apple-1 has no backspace ! For Don Lancaster's original, TTL based TV Typewriter, published in the September 1973 issue of "Radio Electronics" magazine, they soon added circuits for serial I/O and full cursor movement. For more info, see [10].

But back to POM-1. After exiting the typewriter program with the 'ESC' key, you should see a backslash prompt ('\') and a blinking '@' cursor on the next line. You are back to WOZmon, if nothing did crash.

To list the machine language code you just loaded on the screen, type:

```
800.87F <return>
```

and the program will be displayed in hex notation. Note that the last 13 bytes will be zero on the POM-1, because it clears the emulated memory when starting up. The real Apple-1 does not do that, so these bytes may have random contents.

USING ONLINE APPLE-1 EMULATORS

Online Apple-1 emulators such as the one in [12] accept keypresses like the real Apple-1 does. As the contents of the APL and TUR files essentially are “automatic key presses”, just copy the text therein and paste it into the online Apple-1 emulator to load and run your program there. For [12], open its text input window with the LOAD button, and then paste the APL or TUR file into this text window. Close the text window to see the effect.

MAKING MULTI BLOCK PROGRAMS

The normally expected use of this toolchain is to develop small machine language programs of no more than 4 kBytes of object code. This limitation comes from the standard Apple-1 DRAM configuration with 4 kBytes of DRAM in the address range of $0000-$0FFF. The upper 4 kBytes DRAM block at $E000-$EFFF was meant for the BASIC interpreter.

But what if the need arises to use both blocks of memory ?

In this case, you need to split your assembly language program into two blocks, such that two binary files are produced, one for the lower 4k bank, and one for the upper 4k bank.

This requires some additional work to put all the needed equates from one module to another, such that subroutine calls and access to variables do work between the two modules. For the variables it's easy – just "include" the same definitions file in both modules, For the subroutines, it's not so easy – but the task can be greatly simplified by having jump vector sections at the beginning of each block. These have a 'JMP' instruction to every subroutine in that block, so you never need to handle addresses yourself after any program changes moving entry points. Here is the idea:

```
SUB01V JMP SUB01        ← the start of the program block
SUB02V JMP SUB02
        . . .
SUB01  INSTRUCTION(s)
        . . .
       RTS

SUB02  INSTRUCTION(s)
        . . .
       RTS
```

and so on. Whenever you call a subroutine having the label <name> from within another block, use the <name>V label, which you know comes spaced exactly 3 bytes apart from all the others, so it is easy to maintain these jump vector sections manually. Their entry addresses SUB01V, SUB02V, etc., will never change, so no need to edit anything in these tables once they have been set up the first time. This is a well known microcomputer programming technique from back in the 1970s, when RAM memory was so limited that the assemblers running on these machines could only assemble small modules one by one. Keeping the most important subroutine calls like keyboard input and screen output in such a jump table is recommended even for single module programs, as it helps with debugging.

CLEANUP AFTER THE TOOLCHAIN BUILD

Once you have established that the toolchain is fully operational, you can just keep the stuff in your TARGET directory and delete everything else related to the toolchain build which is not needed anymore. I did not provide a script to do this cleanup --- I prefer to keep everything around as modern mass storage is so ridiculously cheap and the sources etc. which were unzipped are so small compared to the terabytes of space available. Note the 'atasm.pdf' file which was automatically put in your TARGET directory. This is "atasm's" manual and will be helpful if you want to use "atasm" efficiently.

OUTLOOK

You now have all the tools needed to develop Apple-1 software in 6502 assembly language at your fingertips. To learn that craft, see [11]. And you don't even need a real Apple-1. Just use the skeleton source as a starting point for your own work. Soon, you will have learned how to interact with the Apple-1 I/O ports directly and you won't need the skeleton anymore. But never forget to place the BEGIN label at the start of your code, and the START label where the program entry point is, to tell the toolchain what to do.

HOW TO CONTACT THE AUTHOR

Try **codenoelppa**.AT.gmail.com --- replace the .AT. with a '@' character, and reverse the order of letters printed in boldface. I hope this small riddle will deter all of the so-called "A.I." systems lurking on the Internet from sending me useless spam emails. And, by extension, don't send me spam emails by yourself, even if you are a real human being. Please do not ask me for fixing bugs in your code – but if you find a bug in my toolchain, or in this documentation, please contact me. Also, if you have constructive suggestions, for this or for future releases of the toolchain, or praise, feel free to send them to me, too.

LITERATURE AND ONLINE RESOURCES

[1] https://www.applefritter.com/content/keyboard-serial-terminal
[2] https://8bitflux.com/#keyboard-serial-terminal
[3] https://apple1software.com
[4] https://8bitflux.com/downloads/keyboard-serial-terminal-manual.pdf
[5] https://www.applefritter.com/content/teaching-keyboard-emulator-cable-new-tricks
[6] https://www.willegal.net/appleii/apple1-software.htm (scroll down to toaiff.c source)
[7] https://github.com/linuxha/atasm
[8] https://sourceforge.net/projects/pom1/files/pom1/1.0.0/pom1-1.0.0.tar.gz/download
[9] https://www.applefritter.com/content/uncle-bernies-apple-1-toolchain-part-1
[10] https://en.wikipedia.org/wiki/TV_Typewriter
[11] https://archive.org/download/6502-assembly-language-programming
[12] https://www.scullinsteel.com/apple1

(end of document)