WP 34s Assembler/Disassembler User Guide

This file is part of ***WP 34S***.

WP 34S is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

WP 34S is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with ***WP 34S***. If not, see http://www.gnu.org/licenses/ .

Table of Contents

Table of Figures

List of Tables

1 INTRODUCTION

The ***WP 34S*** is a "repurposing" project designed for the ***HP-20b Business Consultant*** and ***HP-30b Business Professional*** calculators designed by Hewlett-Packard. These calculators are based on an Atmel ARM 7 single chip device containing 128KB of programmable flash memory.

HP made a serial port available under the back cover-plate of these calculators and through this serial port it is possible to "reflash" the calculator's application software using a software download tool made available by Atmel.

The ***WP 34S*** project team designed new S/W for the calculator chassis to convert the ***HP-20b*** and/or ***HP-30b*** calculators into highly competent Reverse Polish Notation (RPN) scientific programmable calculators (see http://sourceforge.net/projects/wp34s/ for more details). The ***WP 34S*** project was done with the knowledge and occasional assistance of HP staff.

Most programmable calculators have their user programs stored in RAM but, in these modern days, the RAM is backed up by the battery even when the calculator is "off". The ***WP 34S*** has the additional capability of storing user programs in one of 9[1] flash memory pages.

This assembler/disassembler program is targeted at helping to design, archive, distribute, and maintain those ***WP 34S*** user programs.

1.1 Goals

The ***WP 34S*** Assembler/Disassembler was designed to accomplish several goals:

1. Archiving and distribution of ***WP 34S*** programs in a distributable source form (text files) as opposed to binary images.
2. At least in its early days, the ***WP 34S*** design was in flux and the binary op-code map was known to fluctuate from revision to revision. This meant that binary flash images might not be compatible between calculator F/W revisions. By allowing the user programs to be saved in an ASCII source format, they can be more easily ported between revisions.
3. Calculator programmers often develop a suite of programs to accomplish many tasks. The assembler is able to process several source files and concatenate the result into a single flash image allowing some form of primitive library management.

Thus was born the ***WP 34S*** Assembler/Disassembler.

1.2 Notes

This manual may show examples from specific SVN revisions of tools and/or files. It is intended to be representative of the current tool. From time-to-time, the examples may be updated when the clarity of the information needs to be improved. Hence, the fact that the cover picture[2] shows SVN 1173 does not indicate that this is the last version supported.

1.3 Acknowledgements

Without the tireless efforts of Walter Bonin, Paul Dale, and Marcus von Cube (the "Gang of Three"[3]) the ***WP 34S*** would not exist. These guys seem motivated to work around the clock on this device and have answered my questions with a degree of patience that is impressive! Cyrille

[1] The number of flash pages available to the ***WP 34S*** is dependent on the calculator's firmware load. As the capability evolves, this number changes. At time of writing (SVN 1358) this currently stands at 9, but may change in the future – sometimes without notice. Consult the latest documentation for the calculator for up-to-date information.

[2] The cover image was shamelessly stolen from the main ***WP 34S*** documentation. The author begs forgiveness for this and other sins. The author will not be updating the cover image with every new revision of the calculator so the version number shown here is ***not*** indicative of the version of this tool.

[3] Thanks to Gerry Schultz for coming up with a better title than "techo-trinity".

de Brebrisson, Tim Wessman and their cadre of co-conspirators at HP must be mentioned for their foresight in allowing the ***HP-20b*** and ***HP-30b*** calculators a route to flash reprogramming and for providing a wealth of material and information about the machines – including the original emulator and SDK. And finally, Bruce Bergman, who has built a web interface[4] to allow anyone to use this tool, regardless of whether they know what a command line application is or not.

[4] See http://www.wiki4hp.com/doku.php?id=34s:assembler

2 LIST OF ACRONYMS AND DEFINITIONS

CPAN	Comprehensive Perl Archive Network (http://www.cpan.org)
dat	Filename extension of a binary program image from flash or RAM used by the emulator
F/W	Firmware
RPN	Reverse Polish Notation
SDK	Software Development Kit
SVN	Subversion – a revision control tool
S/W	Software
wp34s	Filename extension of a ***WP 34S*** source file (by convention)

3 OVERVIEW

The ***WP 34S*** Assembler/Disassembler uses the op-code format as described by the op-code list generated by the "`calc`" program. See Section 5.1 Generating Reference Op-code List for more details.

The Assembler/Disassembler is currently a single pass, direct translation tool.

The Assembler/Disassembler is written in Perl (version 5.8+) so as to be portable across many platforms.

3.1 Supported Feature Set

- Will assemble one or more ASCII source files into a **WP 34S** binary image.
- Will disassemble a **WP 34S** binary image into an ASCII source listing.
- Can accept alternate **WP 34S** op-code maps to translate source and/or flash images between **WP 34S** SVN revisions.
- Can output a "op-code syntax guide" to help with writing accurate source files.
- Will automatically locate the most current op-code map if updated via SVN.

4 USAGE

This tool is both an assembler and a disassembler in one script. For most people, the disassembler will be the first mode they will use – and this will likely be to disassemble a flash image recovered directly from the ***WP 34S*** calculator and/or the emulator.

The disassembler reads a binary image created with the "SAVE", "PSTO", "P↔", etc. op-codes, or via simply exiting the emulator[5]. These binary images can be generated directly using the emulator or can be downloaded from the calculator using the serial cable, SAM-BA, and instructions in the main ***WP 34S*** documentation.

The Assembler/Disassembler tool is run from a command line shell[6]. For example, to disassemble a set of programs saved from flash page 0, the following command may be used:

```
$ wp34s_asm.pl -dis wp34s-0.dat -o myProgs.wp34s
```

Figure 4.1: Disassembler Example Command-Line

The "-dis" command line "switch" tells the tool to run in disassembler mode. The above command will read the "wp34s-0.dat" binary image[7], detect how many steps are in the one or more programs within the binary file, and print out an ASCII source listing. The ASCII listing is written to a file called "myProgs.wp34s".

For example, suppose the flash image contained a modular normalization program. The output listing, "myProgs.wp34s", might look something like this:

```
// Source file(s): wp34s-0.dat
001 LBL'MOD'
002 RMDR
003 RCL L
004 x[<->]y
005 x<0?
006 +
007 RTN
// 7 total words used.
// 8 total words used.
// 6 single word instructions.
// 1 double word instructions.
```

Figure 4.2: Disassembled Source Example

Though the disassembler prints program "step numbers", these are more for the user's benefit than anything else since they are ignored when the file is assembled. The optional step numbers can also be followed by an optional colon (":") when the user writes their own ASCII source file. Additionally, the LBL mnemonics can be optionally preceded by one or more asterisks. These asterisks on the labels are also ignored by the assembler.

The tool can also take ASCII source files and turn them into binary flash images as well. In order to assemble a source listing into a binary image for the flash, a command similar to following may be used:

---

[5] When exiting, the emulator writes one or more files with the name "wp34*.dat". All these files except the "wp34-R.dat" file may be disassembled with this script. The files named "wp34-[0-3].dat" are flash images and are the primary target file names for assembly output by this tool. See the actual ***WP 34S*** documentation for more (and possibly more correct!) details.

[6] There currently is no GUI, however, some plans may be underway to create such a version.

[7] Conventionally, the flash images have the extension ".dat", however there is nothing that enforces this extension. The ***WP 34S*** emulator, however, recognizes several pre-named ".dat" files. See the main ***WP 34S*** documentation for more details.

```
$ wp34s_asm.pl myProgs.wp34s -o wp34s-0.dat
```

Figure 4.3: Assembler Example Command-Line

The resultant binary flash image, in this case "`wp34s-0.dat`", can be loaded directly into either the emulator or the appropriate flash page in the calculator using instructions found in the main ***WP 34S*** calculator documentation.

The Assembler/Disassembler is located in the "`./trunk/tools/`" directory within the SVN development tree.

4.1 Disassembler Instructions

The disassembler uses a direct lookup technique to translate the series of 16-bit words found in the binary flash image into ASCII source mnemonics. Most ***WP 34S*** op-codes are one 16-bit word, with the exception being those that take a quoted alpha string. The quoted alpha strings are between 1 and 3 characters (eg: `LBL'xxx'`, `INT'yyy'`, etc.). Quoted alpha op-codes take two 16-bit words.

As seen in Figure 4.4, to invoke the disassembler, the tool requires the "`-dis`" switch and the name of the binary image to disassemble[8]. If the "`-o`" switch is not given in disassembler mode, the output is sent to the screen (STDOUT). This can be redirected into a file as well, hence both invocations shown below result in identical listing files.

```
$ wp34s_asm.pl wp34s-0.dat -dis > myProgs.wp34s
$ wp34s_asm.pl wp34s-0.dat -dis -o myProgs.wp34s
```

Figure 4.4: Generic Disassembler Command-Line

The disassembler has several optional parameters that can be used with disassembly mode. The "`-s <# of asterisks>`" switch can be used to prepend a specified number of asterisks to the front of the `LBL` op-codes in the listing. These improve readability of the source, making it easier to locate labels in the listing. If this switch is not present (or if a value of 0 was given), no asterisks will be prepended[9].

```
$ wp34s_asm.pl -dis wp34s-0.dat -s 2 -o myProgs.wp34s
```

Figure 4.5: Disassembler Command-Line For Label Asterisks

For example, suppose the flash image contained a modular normalization program. The output of the command in Figure 4.5 might look something like this:

[8] It should be noted that the order of the command line switches is not important to the assembler/disassembler. Only those switches that take an extra argument are required to be in sequence.
[9] Asterisks in front of the `LBL` op-codes are ignored by the assembler.

```
// Source file(s): wp34s-0.dat
001 **LBL'MOD'
002 RMDR
003 RCL L
004 x[<->]y
005 x<0?
006 +
007 RTN
// 7 total instructions used.
// 8 total words used.
// 6 single word instructions.
// 1 double word instructions.
```

Figure 4.6: Source Example with Label Asterisks

Additionally, the disassembler can be asked to suppress the the step numbers as well. This is sometimes useful when doing text comparisons[10] with previous listings using 'diff-ing' tools[11].

```
$ wp34s_asm.pl -dis wp34s-0.dat -ns > myProgs.wp34s
```

Figure 4.7: Disassembler Command-Line With Step Numbers Suppressed

The above command will produce an output listing that might look something like Figure 4.8.

```
// Source file(s): wp34s-0.dat
LBL'MOD'
RMDR
RCL L
x[<->]y
x<0?
+
RTN
// 7 total instructions used.
// 8 total words used.
// 6 single word instructions.
// 1 double word instructions.
```

Figure 4.8: Source Example with No Line Numbers

The disassembler prints a diagnostic statistic set at the end of the listing which includes the total number of program steps used, the number of single word instructions, and the number of double instructions.

Though the disassembler can print program step numbers, these are only for the user's benefit as they are ignored entirely by the assembler.

4.2 Assembler Instructions

The assembler takes one or more ASCII source listings and translates them into a single flash image. The assembler can be used to “mix and match” various smaller programs into a single flash image creating a “library management” system of sorts.

The assembler enforces the ***WP 34S*** calculator's 506-word limit within any given page of flash memory. Thus, the user must keep the total length of a concatenated set of programs in mind when combining source programs. The statistics printed at the bottom of the disassembly listings

[10] With step numbers present, if a line is inserted, all lines below that line will show up as differences. However, with step numbers suppressed, the one inserted line will show up clearly as the only difference.
[11] Examples of differencing tools include '`diff`', '`tkdiff`', '`meld`', etc.

can be useful for this. If the 506-word limit is exceeded, the assembler will abort with an appropriate error. In this case, the user must re-balance their assembly source file and/or their mix of assembly source files to ensure that the 506-word limit is not exceeded.

Hint: Programs can access alpha labels in other flash pages and/or RAM from the one they are currently running within. Therefore a program or set of programs can be created that use `XEQ` or `GTO` to execute programs in other areas of the calculator. This can be used to increase the effective program size that can be executed.

When assembling a source file, the assembler **requires** that the output flash image filename be named using the "`-o`" switch[12]:

```
$ wp34s_asm.pl myProgs.wp34s -o wp34s-0.dat
// CRC16: 520A
// Total words: 32
// Total steps: 31
```

Figure 4.9: Generic Assembler Command-Line

After a successful assembly run, the assembler will print statistics to the console for the resultant flash image. This will include the CRC16 value, the total words consumed, and the number of steps in the program.

There are a number of options that can be used when assembling a source file as well.

If the user's program does not consume the full 506-word limit within any given flash page, the area beyond the user's program is usually filled with the ***WP 34S*** op-code for "`ERR 03`"[13]. It is possible to program any 16-bit value desired into these unused words by using the "`-f ####`" switch. The numeric value of the "`####`" will be written to these unused words[14].

```
$ wp34s_asm.pl myProgs.wp34s -o wp34s-0.dat -f FFFF
```

Figure 4.10: Assembler Command-Line With Non-Standard Flash Fill

C-style comments can be included in the user's source. Both single line ("`//`") and multi-line comments ("`/* ... */`") are supported.

The source fragment shown in Figure 4.11 will produce an identical binary image to the ones in the previous examples.

[12] The assembler output is a binary file and is not suitable for displaying on the screen with a normal text editor. If the user wishes to see the actual values in this binary file, there are many programs available for this. One common one is:

```
$ hexdump wp34s-0.dat
```

[13] This instruction was chosen by the ***WP 34S*** design team just in case the calculator mistakenly executed program steps from this unused area. The "`ERR 03`" instruction will cause the program to stop and display this error.

[14] An unprogrammed or empty flash word has the value of 0xFFFF so using the following optional command line switch may result in a longer lived flash memory within the calculator, and a quicker programming time: "`-s FFFF`".

```
/* Perform modular reduction with normalization: r = a mod m
   inputs: Y <= a
           X <= m
   output: X == r, such that (0 <= r < m)
*/
***LBL'MOD'
RMDR
RCL L
x[<->]y
x<0?
+     // If negative, add the modulus back in to normalize
RTN
```

Figure 4.11: Assembler Source With Comment Styles

As mentioned in the Disassembler section, though the disassembler writes out program step numbers in its output, these are neither required nor translated during the assembly process[15].

The case and format of the mnemonic **must be exact** with respect to the approved ***WP 34S*** syntax. See Generating Reference Op-code List on page 13 for more details on how to generate a mnemonic syntax guide table.

The assembler can be used to concatenate several ASCII source files into a single flash image. Additionally, more than one program can be within each source file. The only limitation is that the total number of words cannot exceed the 506-word limit.

The following is an example of using the assembler to translate multiple source files into a single flash image.

```
$ wp34s_asm.pl gc.wp34s fp.wp34s mod.wp34s -o wp34s-3.dat
```

Figure 4.12: Assembling Multiple Source Files

This technique can be useful for maintaining a library of source files and mixing and matching various programs into a flash image as required.

[15] It does not matter if some statements include step numbers and some don't, nor does it matter if one or more step numbers are repeated, nor that they are not sequential (nor monotonic, blah, blah, blah). They are just plain ignored during the assembly process.

5 ADDITIONAL INFORMATION

5.1 Generating Reference Op-code List

The ***WP 34S*** Assembler/Disassembler script can create a reference file of legal op-codes as a guideline by using the following command:

```
$ wp34s_asm.pl -syntax legal_opcodes.lst
```

This will produce a rather sizable list of all op-codes[16] the ***WP 34S*** recognizes.

As the ***WP 34S*** evolves, this list may change from time to time. It may be prudent to regenerate the list at intervals as the project progresses[17].

Figure 5.1 shows a fragment of a syntax-helper file[18] (first and last 5 lines).

```
0000  ENTER[^]
0001  CLx
0002  EEX
0003  +/-
0004  .
...
fded  VW[alpha]+[->]D
fdec  VW[alpha]+[->]L
fded  VW[alpha]+[->]I
fdee  VW[alpha]+[->]J
fdef  VW[alpha]+[->]K
```

Figure 5.1: Truncate View of Op-Code Syntax Table

The first field is the hexadecimal value of the op-code used by the ***WP 34S***. The next field(s) are the mnemonic used in the listing. Note that, currently the assembler is not very forgiving of the format of the mnemonic field; it must match the format in this file **exactly**. Where there is whitespace[19] within the mnemonic, the user must retain whitespace in their assembler source files. Many special printing characters are "escaped" by the use of square braces ("`[...]`"). These must also be faithfully reproduced[20].

5.2 Migrating Programs Between Op-Code Revisions

From time to time, the ***WP 34S*** op-code definitions may change because of the addition or deletion of certain functions and/or constants. The combination of being able to extract the flash image, disassemble the program to an ASCII source format using one op-code definition map, and then reassemble it into the new op-code definition map is a useful feature of the ***WP 34S*** Assembler/Disassembler tool.

---

[16] At time of writing, this expanded syntax table is greater than 445KB in size!

[17] A helpful technique is to name the syntax guide file after the SVN revision number of the script source. For example, if the script's SVN number is 1194, a suggestion is to name the syntax guide file "`syntax_1194.lst`".

[18] This file is likely from a ancient version and will likely bare no resemblance to the current version. It is shown to illustrate the table format.

[19] "Whitespace" is a common programming term used to mean one or more contiguous characters that show as spaces. For the purposes of this usage, it is taken to mean one or more tabs and/or spaces in a row.

[20] A planned evolution of the assembler/disassembler project is to make this mnemonic format somewhat more "friendly". However, until then, it is very strict in its mnemonic format.

Using SVN[21] it is possible to generate an op-code definition map for the ***WP 34S*** calculator revision the program was originally running under. This map can be saved and used to disassemble the user's program[22].

```
$ cd <location of svn working directory>
$ svn up -r <REV #>
$ cd ./trunk
$ make
$ ./Linux/calc opcodes > svn_XXXX.op
$ wp34s_asm.pl -dis wp34s-0.dat -opcode svn_XXXX.op > mySrc.wp34s
```

Figure 5.2: Linux Command-Line to Disassemble from a Specified Op-Code Map

Repeat the last step for as many flash images as are required, placing each output listing in a separate "`*.wp34s`" file.

With the working user program(s) safely in a source format, the ***WP 34S*** development branch can be brought up to the head of the SVN tree and the latest op-code map created. The updated op-code map and re-assembly is shown below:

```
$ cd <location of svn working directory>
$ svn up
$ # Observe the SVN number reported at the end of the update.
$ cd ./trunk
$ make
$ ./Linux/calc opcodes > svn_YYYY.op
$ wp34s_asm.pl mySrc.wp34s -opcode svn_XXXX.op -o wp34s-0.dat
```

Figure 5.3: Linux Command-Line to Assemble from a Specified Op-Code Map

Repeat the last step for each flash page required to be translated.

The calculator can now be re-flashed with the latest "`./trunk/realbuild/calc.bin`" image and the newly translated flash user programs restored as per the instructions in the main ***WP 34S*** calculator documentation.

Equally, this technique may be used to move a modern program to an older ***WP 34S*** revision[23]. The user must be careful no new op-codes, constants, and/or conversion factors were used in the modern program when regressing to previous SVN versions. Unrecognized op-codes will cause the assembler to halt and display an appropriate error.

5.3 Automatically Locating Op-code Map

As mentioned above, the script can use either the internal op-code translation map or an external one. The external file can be either specified or automatically "discovered" by the script.

The script decides which op-code map to use in the following order:

1. If the user has referenced a map file via the command line switch (ie: -opcodes svn_1234.op) this map will be used.
2. If the above is not the case, the script looks in the current directory for a file called `wp34s.op`.

[21] It is expected that the user can look up SVN usage via the usual Internet resources so only very bare commands will be presented here – without much explanation.
[22] The Linux command set is presented here. The author has never done these steps on any other operating systems though they *should* be straight forward – provided the library dependencies are satisfied.
[23] Moving modern programs to older ***WP 34S*** releases is seen as very unlikely to happen since the majority of calculators will tend to be re-flashed in the *forward* direction (ie: higher SVN revision numbers).

3. If the local file is not found, the script looks in the directory where the script itself is located for a file called `wp34s.op`.
4. If none of the above is available, the script falls back on its internal op-code map table.

Be aware that the `wp34s.op` file is automatically updated by the design team's SVN build process whereas the assembler script (and its internal table) is not. Hence, the SVN `wp34s.op` file may be more up-to-date than the assembler's internal table with respect to the op-code evolution.

6 EXAMPLES

The "`./library/`" directory distributed with the ***WP 34S*** project includes a number of ***WP 34S*** programs either culled from the calculator's internal XROM database or contributed by users. These programs are intended to be instructive in the use of the ***WP 34S*** assembler source language.

Several variations of source format are presented in the collection of files. These include:

- The various comment formats
- The optional preceding step number
- The optional asterisk(s) on labels

The user is encouraged to use these programs as examples for how to construct ASCII source files for the ***WP 34S***. A README_ASM file is included in the directory with more information.

7 SOURCE OF PERL PACKAGES

Perl is a platform independent interpreted scripting language. Perl interpreters are available for many different O/S platforms.

7.1 Linux

A Perl interpreter is included with virtually every Linux distribution available[24]. With Linux, there is usually nothing more that needs to be done beyond running the examples as shown in the preceding sections.

7.2 Windows

Perl must be added separately to a Windows system. Fortunately there are several very good Perl packages available for Windows[25].

There are three free packages that are particularly recommended – none above the other, and in no particular order.

7.2.1 Cygwin

Cygwin is a package intended to provide many of the Linux command utilities in a Windows environment. It is available from the http://cygwin.org/ website. Installation of cygwin is (well) beyond the scope of this document – and is sometimes not trivial to complete – but is well worth the effort if you wish to enjoy a wide variety of Linux-like commands.

Once installed, it is best to open a command shell (bash, zsh, csh, etc.) and "fill your boots"[26]. If you are well versed in Linux environments, cygwin will be immediately familiar.

7.2.2 Strawberry Perl

Strawberry Perl is a Perl package complete with a full collection of Perl libraries. It is available as a Windows install package from the http://strawberryperl.com/ website.

Since this package's installer does not make the pathext associations during installation, in order to run a Perl script, the user must "give the script" to the Perl interpreter for it to be run. For example, Figure 7.1 shows how this is done:

```
$ perl ..\trunk\tools\wp34s_asm.pl -dis wp34s-0.dat > mySrc.wp34s
```

Figure 7.1: Running Script Under Strawberry Perl

The Strawberry Perl installer does not setup the file associations for Perl scripts (as ActiveState does). There are many good reference for "correcting" this oversight[27].

7.2.3 ActiveState Perl

ActiveState Perl is another free Perl package distributed by a commercial venture that also supplies a licensed enterprise version. This is available from the http://www.activestate.com/activeperl website.

ActiveState Perl has a slight advantage over Strawberry Perl in that its installation script will set appropriate associations to the '`.pl`' extension for the Perl interpreter, making the command shell

[24] Actually, the author is unaware of any that does *not* include a Perl package.
[25] And some very poor packages as well (*caveat utilitor*!).
[26] It is not the intention of this document to teach the finer points of cygwin usage. Consult one of the many fine on-line documents for further information.
[27] See http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/ftype.mspx for an example of how this may be done.

automatically recognize Perl scripts and launch the interpreter accordingly. Hence, other than the fact that the path names must be "Window-ized", the examples shown in the first part of this manual are equally valid.

```
$ ..\trunk\tools\wp34s_asm.pl -dis wp34s-0.dat > mySrc.wp34s
```

Figure 7.2: Running Script Under ActiveState Perl

7.2.4 Native Windows Executable

There is an experimental Windows stand-alone EXE file built using a Perl-to-executable module called PAR::Packer[28]. PAR::Packer includes a tool called "pp" which "packages" the Perl script along with Perl libraries into an executable form[29].

This assembler/disassembler has been packaged using the "pp" tool into a Windows executable.

```
$ wp34s_asm.exe -dis wp34s-0.dat -o mySrc.wp34s   # disassemble
$ wp34s_asm.exe mySrc.wp34s -o wp34s-0.dat        # assemble
```

Figure 7.3: Running the Windows Executable

The executable should be usable on virtually any Windows installation with no additional libraries or files required[30]. It has exactly the same command line usage as the base Perl script – albeit with the requirement to "Window-ized" the directory paths.

A few words on the "pp" tool and its output:

1. The executable image is somewhat on the *large* size because of the way "pp" lumps into the EXE many libraries which may not be strictly required. A quick stab at reducing the footprint has been attempted (this succeeded in reducing the "static" executable footprint from ~12.5MB down to ~5.6MB!). If time permits, more work will be done to further reduce the footprint.
2. Since "pp" constructs a compressed image of the Perl libraries, the first invocation will take some time to decompress the image. Subsequent runs should benefit from the fact that "pp" mysteriously caches the decompressed image (don't ask me how, don't ask me where!).
3. Since the decompressed image must exist for the script to run, the "dynamic" footprint (ie: the decompressed directory) will be somewhat larger than the static executable. The author is not sure exactly how large this is but will estimate in the 20MB+ range (purely a guess!).

7.3 MAC O/S

The author is under the impression that MAC O/S comes with a Perl interpreter. However, he is not actually very familiar with the MAC. If someone would care to update this section, it would be appreciated!

[28] Available from CPAN. Current version v1.009.
[29] At time of writing, the native executable version of the script was built using the aforementioned version of PAR::Packer and Strawberry Perl, version 5.12.1.0.
[30] The executable version of the script will also automatically locate the `wp34s.op` file, if available, as described in Section 5.3.

8 HELP

The assembler/disassembler has a help screen which explains the command-line arguments that can be used. Figure 8.1 shows the help screen circa SVN 1187. This may change with later revisions. However, the one below is expected to be reasonably representative of later versions as well. Use the '-h' switch to display this help screen. The native executable version will show slight and unimportant differences.

```
$ wp34s_asm.pl -h
wp34s_asm.pl  - Assembler/Disassembler for the WP34s calculator.

Usage:
   wp34s_asm.pl src_file [src_file2 [src_file3]] -o out_binary  # assembly mode
   wp34s_asm.pl in_binary -dis > src_file                       # disassembly mode

Parameters:
   src_file         One or more WP34s program source files. Conventionally, "wp34s" is used
                    as the filename extension.
   -o out_binary    Flash image produced by assembler. Required in assembler mode. Conventionally,
                    "dat" is used as the filename extension.
   -dis             Disassemble the binary image file.
   -op file         Optional opcode file to parse.              [Default: --internal table--]
   -fill fill_hex   Optional value to prefill flash image with.  [Default: instruction 'ERR 03']
   -s number        Optional number of asterisks (stars) to prepend to labels in
                    disassembly mode.                           [Default: 0]
   -ns              Turn off step numbers in disassembler listing.
   -h               This help script.

Examples:
  $ wp34s_asm.pl great_circle.wp34s -o wp34s-3.dat
  - Assembles the named WP34s program source file producing a flash image for the WP34s.

  $ wp34s_asm.pl  great_circle.wp34s  floating_point.wp34s  -o wp34s-1.dat  -fill FFFF
  - Assembles multiple WP34s program source files into a single contiguous flash image for
    the WP34s. Uses FFFF as the optional fill value. Allows (and encourages) use of libraries
    of programs by concatenating the flash image from several source files.

  $ wp34s_asm.pl -dis wp34s-1.dat -s 3
  - Disassembles a flash image from the WP34s. Prepend 3 asterisks to the front to each label to
    make then easier to find in the listing (they are ignored during assembly).

  $ wp34s_asm.pl -dis wp34s-0.dat > test.wp34s ; wp34s_asm.pl test.wp34s -o wp34s-0a.dat
  $ diff wp34s-0.dat wp34s-0a.dat
  - An end-to-end test of the tool. Note that the blank fill mode will have to be the same
    for the binaries to match.

Notes:
  1) Step numbers can be used in the source file but they are ignored. Since they are ignored,
     it doesn't matter if they are not contiguous (ie: 000, 003, 004) or not monotonic (ie: 000,
     004, 003). The disassembler does produce step numbers that are both contiguous and monotonic.
  2) You can name a different opcode table using -opcode. This can be used to translate a source
     written for a different SVN revision of the WP34s to move the program to a modern version of
     the WP34s. Disassemble the old flash using the old opcode table and reassemble using the default
     (internal) table. This is also an insurance policy against the opcodes evolving as well. Simply
     target newer opcode tables as they become available. To generate an opcode table, using the
     following (Linux version shown, Windows is likely similar):
      $ svn up
      $ cd ./trunk
      $ make
      $ ./Linux/calc opcodes > new_opcodes.map
      $ wp34s_asm.pl -dis wp34s-0.dat -opcode new_opcodes.map > source.wp34s
  3) The prefill-value will be interpreted as decimal if it contains only decimal digits. If it
     contains any hex digits or it starts with a "0x", it will be interpreted as a hex value. Thus
     "1234" will be decimal 1234 while "0x1234" will be the decimal value 4660. Both "EFA2" and
     "0xEFA2" will be interpreted as a hex value as well (61346). The leading "0x" is optional in
     this case.
  4) The order the command lines switches are used is not important. There is no fixed order.
```

Figure 8.1: wp34s_asm.pl Help Screen