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 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). This 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 4 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 a source format, they can be easily ported between revisions.
3. Calculator programmers often develop a suite of programs to accomplish many tasks. The assembler is able to process several 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 to current tool. From time-to-time, the examples may be updated when the clarity of the information needs to be improved.

2 LIST OF ACRONYMS AND DEFINITIONS

`dat`	Filename extension of a binary program image from flash or RAM used by the emulator
F/W	Firmware
RPM	Reverse Polish Notation
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 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 map” to help with writing accurate source files.

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 disassembler a flash image recovered directly from the ***WP 34s*** calculator and/or the emulator.

The disassembler reads a binary flash image created with the "SAVE", &&& op-codes. These binary flash 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 tool is run from a command line shell[1]. For example, to disassemble a set of programs saved in flash page 0, the following command may be used:

```
$ wp34s_asm.pl -dis wp34s-0.dat > 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[2], detect how many steps are in the one or more programs within the binary file, and print out an ASCII source listing.

For example, suppose the flash image contained a modular reduction 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: Conventional Source Example

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

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:

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

Figure 4.3: Assembler Example Command-Line

[1] There currently is no GUI, however, some plans may be underway to create such a version.
[2] 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 form more details.

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.

4.1 Disassembler

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 4.1, to invoke the disassembler, the tool requires the "`-dis`" switch and the name of the binary image to disassemble[3].

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

Figure 4.4: Generic Disassembler Command-Line

The disassembler has several optional parameter 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[4].

```
$ wp34s_asm.pl -dis wp34s-0.dat -s 2 > 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:

```
// 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 with previous listing using 'diff-ing' tools[5].

[3] It may be noted that the order of the command line switches is not important to the assembler/disassembler.
[4] As previously noted, asterisks in front of the `LBL` op-codes are ignored by the assembler.
[5] Examples of differencing tools include '`diff`', '`tkdiff`', '`meld`', etc.

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

Figure 4.7: Disassembler Command-Line With Suppressed Step Numbers

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

4.2 Assembler

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 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 running within. Therefore a program or set of programs can be created that use XEQ or GT0(?) 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[6]:

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

Figure 4.9: Generic Assembler Command-Line

[6] The assembler output is a binary file and is not suitable for displaying on the screen. 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
```

After a successful assembly run, the assembler will print out statistics 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`"[7]. 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[8].

```
$ 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 the single line ("`//`") and multi-line comments ("`/* ... */`") are supported.

```
/* Perform modulus 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 Comments Styles

The above source fragment will produce the identical binary image to the ones for the previous examples.

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[9].

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 11 for more details on how to generate a mnemonic syntax table.

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

[7] This instruction was chosen by the ***WP 34s*** design team just in case the calculator mistakenly executed from this unused area. The "`ERR 03`" instruction will cause the program to stop and display this error.
[8] 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`".
[9] It does not matter if some statements include step numbers and some don't, nor does it matter if the step numbers are repeated, nor that they are not sequential. They are plain ignored during assembly.

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[10] 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.

Figure 5.1 shows a fragment of the syntax-helper file (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[11] 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[12].

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. The combination of being able to extract the flash image, disassembler the program to a source format using one op-code definition map, and reassemble into the new op-code definition map is a useful feature of the ***WP 34s*** assembler/disassembler tool.

Using SVN[13] it is possible to generate an op-code definition map for the ***WP 34s*** calculator revision the program was originally running under. This can be saved and used to disassemble the working program[14].

---

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

[11] "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.

[12] An 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.

[13] 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.

[14] The Linux command set is presented here. The author has never done these steps on any other operating systems though they *should* be quite straight forward – provided the library dependencies are satisfied.

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

Figure 5.2: Linux Command-Line to Create Old 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 program(s) safely in a source format, the ***WP 34s*** program can be brought back to the head of the SVN development 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 > svn_YYYY.op
$ wp34s_asm.pl mySrc.wp34s -opcode svn_XXXX.op -o wp34s-0.dat
```

Figure 5.3: Linux Command-Line to Create Updated 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 image and the translated flash 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[15]. The user must be careful no new op-codes, constants, and/or conversion factors were used in the modern program. Unrecognized op-codes will cause the assembler to halt display an appropriate error.

[15] Moving modern programs to older ***WP 34s*** releases is seen as very likely to happen since the majority of calculators will tend to be re-flashed in the forward direction (ie: higher SVN revision numbers).

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 an 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[16]. 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[17].

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"[18].

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 Perl Under Strawberry Perl

7.2.3 ActiveState Perl

ActiveState Perl is another 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 the previous two in that its installation script will set associate the '`.pl`' extension for the Perl interpreter, making the command shell automatically recognize Perl scripts and launch the interpreter accordingly. Hence, other than the path names must be "Window-ized", the examples shown in the first part of this manual are valid.

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

Figure 7.2: Running Perl Under ActiveState Perl

[16] Actually, the author is unaware of any that does *not* include a Perl package.
[17] And some very poor packages as well (*caveat utilitor*!).
[18] 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.

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 appreciate!

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, this is expected to be representative of later versions as well. Use the '-h' switch to display this help screen.

```
$ 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   Optinal 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 does'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 continguous 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