Using the m1pp macro preprocessor

This document describes the m1pp preprocessor and how to use it

What is m1pp?

M1pp is a preprocessor for macro and macro1 assembler files. It has two functions.

First, it checks for some common errors and either fixes them or gives warnings.

Second, it adds several new features.

The processed output is fully compatible with both macro and macro1.

The checks

- A missing label as the first line
- A comment after code with only tabs separating it
- Text following a location directive
- Shifted characters in a flexo pseudo-instruction
- Combined instructions vs instructions separated by tabs
- Constants seen but no constants statement
- No start at the end of the program
- Extra text after a start

The missing label test does a simple check for the first line starting with digits followed by a forward slash, or a sequence of alpha characters followed by a comma. Only a warning is given, the line is preserved as-is.

A comment on a line after code with only tabs separating it causes both assemblers to give an error. When detected, the comment is moved to the following line.

Text following a location directive, e.g., '100/ xxx' silently produces bad code. Everything following the slash is ignored, so something like '100/ sza' loses the 'sza'. When detected, the rest of the line following the slash is moved to the next line.

Shifted characters in a flexo pseudo-instruction warns about the broken behavior of flexo when using shifted characters. A character that causes a shift change results in two characters, the shift code, 072 or 074, and the actual character. This means that *flexo Abc* will result in *074 A 072*, not what what was expected. Flexo assumes it begins in unshifted mode, so unshifted characters work as expected. This is another silent failure.

Instructions on the same line can be separated by spaces or tabs. Seeing the difference can be difficult. Spaces mean 'add together', tabs mean 'start a new line'. For clarity, if tabs are seen between instructions on the same line, each will be moved to a separate line.

The ideal check would be to determine if adding the words makes sense. For example, 'idx x jmp y' is almost certainly meaningless and would result in a single totally different instruction than intended. However, this version doesn't check for that, it would require parsing the code as if it was being assembled. So, it is still possible to have nonsense

code.

If one or more constants, (xxx, have been seen but there is no following constants directive, a warning is given. It is legal to have more constants following the directive as long as there is always a directive to emit them before the end of the program.

If there is no start statement at the end of the program, a warning will be given. If there are more lines following the start statement, a warning will be given because the assembler does not process those lines.

New features

Some convenience features have been added. These are nonstandard and if you use them in your code, you must run it through m1pp. If that is done, the generated code is fully compatible.

Why add them? Because they're useful, the ascii extensions particularly so for the SCS socket communication IOT. The include capability makes it easy to have 'library' code that can be inserted into a program easily. If you object and don't want to use them, then just don't use them.

The features are:

- A hexadecimal number of the form *0xn*, the standard C/C++ representation, can be used anywhere a number can be used. It will be converted to either an octal or decimal number, depending upon the current radix that is set.
- A literal ascii character of the form '*x*' can be used anywhere a number can be used. The usual escape sequences are recognized. It will be converted to either an octal or decimal number, depending upon the current radix that is set.
- A new pseudo-instruction *ascii* "*text*" is added. When used, it works just as the macro *text* pseudo-instruction works, except it generates the numeric equivalent of each ascii character, again depending upon the current radix. Characters are packed two per word with the first being in bits 2-9, the second in 10-17. The two high bits will be zero. The final character will always be a byte of 0, the usual string terminator. A text string is limited to no more than 1024 bytes.
- Include file support is added. The statements *#include* "*file*" or *#include* <*file*> appearing at the beginning of a line by itself will insert the contents of the file at that point, just as would occur in C or C++. Any text following the closing quote or bracket is added as a comment on a line by itself. Nested includes are supported, up to 1024 files deep, which admittedly is a bit excessive. The default root location when <*file*> is used is */opt/pidp1/MacroIncludes*, but see the usage details below.

Building m1pp

Just type make.

Usage

m1pp [-o outfile] [-e errfile] [-S sysrootpath] [sourcefile]

The -S sysrootpath directive changes the location that *#include* <*x*> files are searched for.

If no input file is given, stdin is used.

If no output file is given, stdout is used.

If no error file is given, stderr is used.