Two pass Assembler (System Software Course Project)

Compiling and running the project

To compile the project, use the *makefile* provided in the project root directory. This will generate the “assembler” executable file. Input and output files must be provided in order to run the program. To start the executable, use the following command (filenames must be provided in the following order):

```
./assembler -o output.txt input.s
```

The output file should have either a .txt or .o extension

Implementation details

The parser reads the input file, divides it into lines and forwards the lines to the assembler.

The most important flags and counters used during compilation are:

- `location_counter`: tracks the number of bytes used so far
- `line_counter`: tracks the number of lines processed
- `token_counter`: marks the current token in the current line

Location counter is reset each the compiler enters a new section or starts a new pass. It is increased every time a program allocates memory or uses an instruction, by the number of bytes used. Token counter is reset every time the compiler moves to the next line and is incremented by one when the processing of the current token is done.

The assembler has two main functions: first and second pass. During the first pass, the symbol table is formed and during the second pass the compiler generates relocation data and object output (machine code). Both passes follow this general algorithm:

1. Traverse through the input file
2. Don’t process empty lines
3. Tokenize the current line (during tokenization all whitespace, commas and code comments are removed)
4. Check if the current line contains a label; if it does, make a new symbol table entry
5. Check if the current line contains a directive; if it does call, the corresponding handler
6. Check if the current line contains an instruction: if it does, call the corresponding handler
7. Check if the compilation is done; if it is not, go to step 1 and repeat

A directive handler’s only task is to recognize if the current token is a directive, and what type of directive is it and then to call the specific directive handler, if needed. Each type of directive has its own handler. The same goes for instructions.

After the compiler has finished, the function `print_data()` is called and the symbol table, relocation data and object file output it written to the output file.