# Code documentation for computer organization project

## **Assembler**

General logic:

1. Pass through the assembly source code – In this pass we only check at which lines there are labels and update **g\_labels\_arr**.
2. Rewind to the beginning of the source code file.
3. Pass through the assembly source code – In this pass we:
   1. Read each assembly line, decode it to hexadecimal and write it to **imemin.txt** file.
   2. For each **.word**  command we update **g\_data\_memory**
4. Write **g\_data\_memory** to **dmemin.txt** file.
5. Close all the open files.

Part 1:

Runs **pass\_over\_file** function.

First the function checks:

1. If the line is empty by checking if the first letter is '\n'
2. If the line is a comment by calling **is\_line\_comment**

If 1 or 2 is true then we skip this row, otherwise we check if the line has a label in it. If the line does contain a label, then we create a new **label\_t** object with:

* label = the label we just got
* cmd\_index = **g\_command\_counter** (This is a global counter that counts how many of the lines we read are real commands)

Now we add the new **label\_t** to the next empty spot in **g\_labels\_arr**.

Part 3:

In this part we use the same function as in part 1, the only difference is that we check if a line is **.word** command and if so we update **g\_data\_memory**. We also check if a line is a real command and if so run **decode\_cmds\_to\_output\_file** to decode it to hexadecimal and write it to **imemin.txt**.

Helper functions:

**get\_opcode\_num** – Searches for the opcode's index in **opcodes\_arr**.

**get\_regs\_num** – Searches for the register's index in **regs\_arr**.

**get\_label\_num** – Searches for the label's index in **g\_labels\_arr**.

**does\_line\_contain\_label** – checks if line contains ':'

**is\_label** – gets immediate value and checks if the first char is a letter or not (to know if the immediate gets a value or a label)

**decode\_cmds\_to\_output\_file** – parse the line and uses the previous codes to get the hexadecimal decoding.

**add\_data\_to\_memory** – use sscanf() to get the value and address for **g\_data\_memory**, also update **g\_max\_memory\_index** for when we write the file (so we won't write the empty memory).

**write\_memory\_file** – simple for loop.

**clear\_leading\_white\_spaces** – using isspace() to skip white spaces.

**is\_line\_comment** – checks if the first letter is '#' (we already cleared white spaces at this point).

**line\_has\_label** – returns the index of ':' or -1 if there isn't ':' or there is '#' before it.

**is\_line\_word\_command** – checks if the first letter is '.' (we already cleared white spaces at this point).

**line\_has\_command** – checks if the first word in the line (after the label) is a valid opcode using **get\_opcode\_num**. If it is then the line contains a command, otherwise it doesn't.

## **Simulator**

General logic:

1. Loading input files
2. Executing assembly commands
3. Writing output files
4. Close files