# Report for Assignment 1

## big picture thoughts and ideas

1. The key role of label table:

   To store the labels is the first thing that happens in assembler's job. This is because that besides executing row by row, other ways of procedure control is **branch** and **jump**, which takes great advantage of stored instruction address. And the only thing that a assembler cannot recognize is labels named by users. Only after implementing a label table, the assembling process can happen smoothly.

2. translation to binary number:

   The whole assembling process is about substituting registers, address, operations and immediate value with binary representation of proper length. Thus sufficient and accurate conversion is required. And it may be dangerous when dealing with address.

3. dealing with the whole .asm file:

   A .asm file has many sections and many user-defined labels. Thus forms of input will be various because of indent. Project  needs to be designed robust enough to handle different files.

4. steps:

   | step1 | step2 | step3 |
   |---|---|---|
   | implement a label table | change each line of instructions into a processible form | translate each line and  output result to specific file |

## The high level implementation ideas

1. **step1**:
   - step 1.1

     define a data structure `LabelTable` that can store the relation between label and the address it represents

   - step 1.2

     read through the whole file line by line

     add 4 to program counter if the line has a instruction

     store the address of next instruction if the line has a label

2. **step2**:
   - step 2.1

     define a data structure `InstructionPartVec` that stores different parts of a instruction

   - step 2.2

     break each instuction into parts, skipping *white spaces* and commas

3. **step3**:
    - step 3.1

      find the type of the instruction according to the first part of the instruction
    - step 3.2

      translate into binary representations using mapping between mips and machine codes
    - step 3.3

      output to target file

# The implementation details

## data structures

1. `LabelTable` in labelTable.hpp

   ```
   typedef std::unordered_map<std::string, unsigned int> LabelTable;
   ```

   uses unordered_map to store the label as a `string` and the address as a `unsigned int`, which is efficient.
2. `InstructionPartVec` in phase2.hpp

   ```
   typedef std::vector<std::string > InstructionPartVec;
   ```

   uses `std::vector<std::string >` to store parts of instructions as strings. For example:

   `add rd, rs, rt` stored as `add` `rd` `rs` `rt`

   `lw rt, immediate(rs)` stored as `lw` `rt` `immediate(rs)`

## functions

1. `LabelTable phase1(std::string inputFilePath);`

   is a entrance of calling **phase1** of assembling process, i.e., creating a label table.
2. `void read_labels(std::string inputFilePath, LabelTable & labelTable);`

   reads through the file and create a label table.
3. `std::string int_to_bin_str(int PC, int bitLength);`

   translates a integer `PC` into corresponding binary representation string of length `bitLength`.
4. `char what_type(std::string instruction);`

   tell what type of mips instruction it is.
5. `std::string get_offset_str(std::string offsetAndBase);`

   get the offset in the form "offset(base)".
6. `std::string get_offset_str_branch(unsigned int PC, unsigned int target);`

   calculate the offset for branch instructions and return the result in binary representation string

7. `std::string get_base_str(std::string offsetAndBase);`

   get the base in the form "offset(base)"

8. `std::string translate(std::string line);`

   translate a line of instruction, pass splited instruction into `translate_R/I/J`.

9. `std::string translate_R(InstructionPartVec parts);`

   translate an R instruction into binary form

10. `std::string translate_I(InstructionPartVec parts);`

    translate an I instruction into binary form

11. `std::string translate_J(InstructionPartVec parts);`

    translate an J instruction into binary form

12. `void phase2(std::string inputFilePath, std::string outputFilePath, LabelTable l);`

    called to start **phase2**