# ECE 2500 Project 1

**Total points:** 1000

**Version:** 1 (Some minor updates might be needed. They will be announced through announcements, so you will not miss them. )

**Deadline:** Friday, Oct. 2nd, 9:00 PM (Refer to Late policy from syllabus in the below paragraph.)

**Late Policy:**Projects have strict deadlines. They have to be digitally submitted at 9 PM on the day they are due. If they are submitted before 10 PM, 5% reduction in grade will apply and the project will still be accepted. If they are submitted by 9 PM of the day after due date, 20% reduction in grade will be applied and still the project will be accepted. No project will be accepted after that point without a letter from Dean’s office. In order for your projects to be accepted, you need to complete the submission process. Uploading the files but not making it available to the instructor, submitting the wrong version, and other technical or non-technical issues do not make you eligible for a late submission.

**Note1**: For this project, we will use Moss (a System for Detecting Software Plagiarism) from Stanford University to detect similarities between submitted projects for this year and previous years. Any similarities beyond what is expected due to using of the starter code will be carefully examined. Suspicious submissions will be reported to Honors committee for further analysis.

**Note 2**: For this project, you are allowed to use C++ standard libraries as well as Qt library. This means if you happen to need any data structure such as map and linked list, you do not need to implement them yourself.

### Project description:

For this project, you will build an assembler using C++ code. The input to the assembler will be a text file named \*.asm, where \* is the name of an assembly file in the same folder as the assembler executable. The output of the assembler is another text file, which will be called \*.txt (same name but different extension). The executable will be named myAssembler. The executable will be called from the console as follows:

myAssembler \*

In such case, myAssembler will read \*.asm and generate \*txt.

Your grade will be based on the correctness of the object code in \*.txt. You can use the console to print out messages for debugging purposes. The summary of myAssembler capabilities are listed below:

* The input of myAssembler is a text file that contains MIPS assembly instructions along with comments.
* The output of myAssembler is a text file that contains one line for each MIPS instruction in the assembly code. Each line will contain the assembled instruction in hexadecimal characters.
* myAssembler should be able to assemble all the instruction in the core instruction set with the exception of j and jal instructions. The core instructions are summarized in the top left table of MIPS reference card, also known as Green Card.
* myAssembler should be able to disregard any comment in the assembly code, after it encounters a # character
* myAssembler should be able to print an error message and exit, if it encounters an error in the assembly code or finds an instruction that cannot assemble. The error message should say “Cannot assemble the assembly code at line …(fill with line number)”
* In the case of bne and beq instructions, if the target label is not defined in the assembly file, myAssembler should print an error message and exit. The error message should say “Undefined lable …. (fill with lable) at line …(fill with line number)

(NOTE: There will be at most one label in each line of the program)

* myAssembler should be able to assemble immediate numbers, both in hex or decimal format.

The following table shows how the grade is distributed.

|  |  |
| --- | --- |
| The skeleton of the assembler, which is able to assemble R-type instructions | 500 points |
| I-type arithmetic and logical instructions | 100 |
| Branch instructions | 300 |
| Documentation and 1-page report | 100 |

The code should be sufficiently documented. A 1-page report should give a high-level description of your solution including algorithm and data structure. If needed, you can add a maximum of 1 page to your report.

Hint for debugging: In order to make sure you have created the correct object code, you can get help from QtSpim. When you load an assembly file into QtSpim, it gets assembled and loaded into the simulator memory. You can see the object code in hex format under the Text tab, in the second column of the table.

When in doubt about how your assembler should behave for certain situations, use QtSpim’s behavior as a guide.

In the QtSpim Simulator setting, you need to check “Enable Delayed Branches”. Otherwise, the numbers you expect to be encoded as the immediate for the target would be different from lecture notes.

**Starting Kit**

An initial code setup has been provided to you under the Project 1 link. You can create a Visual Studio project by adding the appropriate source and header files provided. The code essentially reads an ‘.asm’ file and parses all the instructions in it. For example consider the following piece of code

Start: add $t1, $t2, $t3

sw $t1, 0($t2)

The provided code will parse the file using lexer.cpp and return a vector of ‘instructions’ (instruction is a struct declared in the code). Each instruction has the following fields:

1. labels (If there are any labels in the current instruction line)
2. name (name of the instruction)
3. args (vector containing the register names, immediate values etc)

For the above example, the parser output will be

Instruction 1 :

{

labels = ‘Start’

name = ‘add’

args = {‘$t1’, ‘$t2’, ‘$t3’}

}

Instruction 2 :

{

labels = NULL

name = ‘sw’

args = {‘$t1’, 0, ‘$t2’}

}

There are 3 test files (.asm) provided on the project 1 link along with the desired outputs (.txt) for them. Try running the provided code for these test cases and you will see the parsed code on the console window (screenshot below). You do not need to understand the lexer.cpp code. You can just treat it as a black box. ***You can build your code on top of the provided setup or you can start from scratch, if you prefer.***

Please note that your final submission will be run on similar test cases so you have to follow the input and output protocols exactly.

