**CSE26101 Project 1: Implementing a MIPS Assembler**

***Due 11:59PM, Oct 30th***

1. **Overview**

This project is to implement a MIPS (subset) ISA assembler. The assembler is the tool that converts assembly codes to a binary file. The goal of this project is to help you understand the MIPS ISA instruction set and be familiar with the principle of assemblers.

This assembler is a simplified assembler that does not support the linking process, and thus you do not need to add the symbol and relocation tables for each file. In this project, only one assemble file will be the whole program.

You should implement the assembler that can convert a subset of the instruction set shown in the following table. In addition, your assembler must handle labels for jump/branch targets, and labels for the static data section.

1. **Instruction Set**

The detailed information regarding instructions are in the attached MIPS green sheet page.

| ADD | ADDI | ADDIU | ADDU | AND | ANDI | BEQ | BNE | J | JAL |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| JR | LHU | LUI | LW | LA\* | NOR | OR | ORI | SLT | SLTI |
| SLTIU | SLTU | SLL | SRL | SH | SW | SUB | SUBU | MOVE\* |  |

* Only loads and stores with 4B word and halfword need to be implemented.
* The assembler must support decimal and hexadecimal numbers (0x) for the immediate field, and .data section.
* The register name is always “$n” n is from 0 to 31.
* la (load address) is a pseudo instruction; it should be converted to one or two assembly instructions.

la $2, VAR1: VAR1 is a label in the data section

➜ It should be converted to lui and ori instructions.

lui $register, upper 16bit address

ori $register, lower 16bit address

If the lower 16bit address is 0x0000, the ori instruction is useless.

Case1) load address is 0x1000 0000

lui $2, 0x1000

Case2) load address is 0x1000 0004

lui $2, 0x1000

ori $2, $2, 0x0004

* move is a pseudo instruction; it should be converted to a specific assembly instruction.

move $1, $2

➜ It should be converted to add instruction with $0 as a **target register(rt)**.

**2.1 Directives**

.text

- indicates that following items are stored in the user text segment, typically instructions

- It always starts from 0x400000

.data

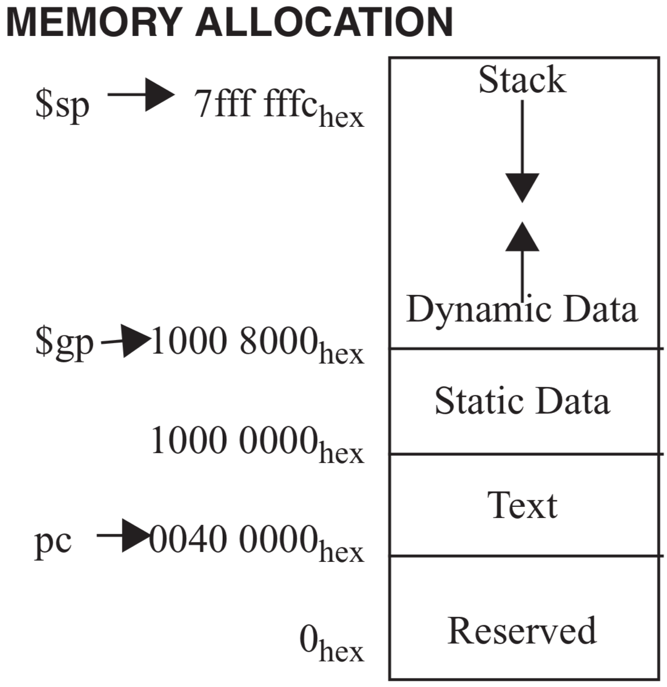
- indicates that following data items are stored in the data segment

- It always starts from 0x10000000

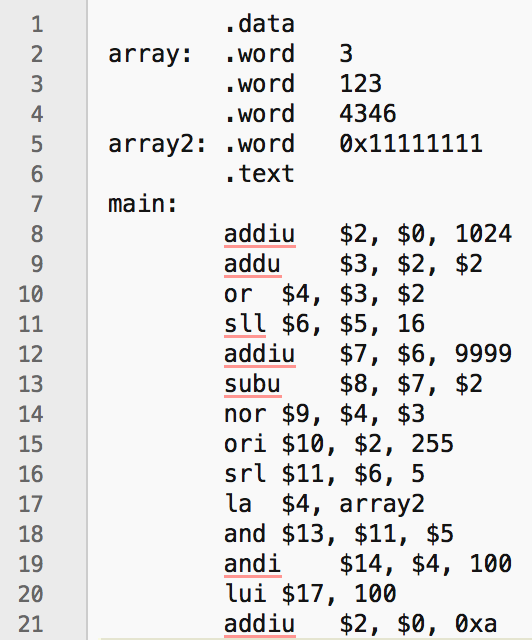
.word

- store n 32-bit quantities in successive memory words

You can assume that the .data and .text directives appear only once, and the .data must appear before .text directive. Assume that each word in the data section is initialized (that is, each word has an initial value). In the following figure, we illustrate the memory map used in our projects.



**2.2 Input format**



Here is one of the input files we will use. As mentioned in Section 3, each input file consists of two sections, data and text. In this example, array and array2 are data.

**2.3 Output format**

The output of the assembler is an object file. We use a simplified custom format.

- The first two words (32bits) are the size of text section and data section.

- The next bytes are the instructions in binary. The length must be equal to the specified text section length.

- After the text section, the rest of bytes are the initial values of the data section.

The following must be the final binary format:

<text section size>

<data section size>

<instruction 1>

…

<instruction n>

<initial values of the data section>

1. **Download Project1 Repository**

You can download the skeleton code from the 2022-Fall-Computer-Architecture/uni{student ID}/Project1 repository to server or local machines. Then you are ready to start the project.

1. Go to each gitlab page. The page will contain project1 directory.
2. Change directory to the location you want to clone your project and clone!
3. You will get the clone repo in your machine.

If you do not want to use the skeleton code, you are allowed to write code from scratch. However, you are supposed to follow the input and output file format because the grading script works on the provided sample\_input and sample\_output files described in the following section.

1. **Grading Policy**

You can test your assembler by using examples in the sample\_input directory. Your assembler should print exactly the same output as the files in the sample\_output directory.

You are encouraged to use the diff command to compare your outputs to the provided outputs. If there are any differences (including whitespaces) the diff program will print the different lines. If there are no differences, nothing will be printed. Furthermore, we have provided a simple checking mechanism in the test.sh. Executing the following command will automate the checking procedure of all test cases.

$ ./test.sh

Sample data and the data for grading may be different. So even if your assembler works correctly for all sample data, you may not get a perfect score. Your assembler should work correctly for all possible data, not just the given sample data.

Each test case is scored out of 10, and each is scored in the following way:

1) Compare the output with the correct answer through diff command to get the number of different lines\*.

(\* output of diff -y --suppress-common-lines <file1> <file2> | wc -l)

2) Deduct 1 point for each different line.

3) If there are more than 10 different lines, 0 points are given.

1. **Submission**

The grading will be on the code at **Master branch** at the deadline. Any other branches are not considered as submission of the assignments. Please make sure that you can see the same result on the submission on the master branch as on your local machine.

1. **Updates/Announcements**

If there are any updates to the project, including additional tools/inputs/outputs, or changes, we will post a notice on the BB and Piazza. Please check the notice for any updates.

One concern is that as we are using Python as an assignment language, there are plenty of packages you can leverage. Therefore, we FORBID you from importing external packages. If you have a question about using an external package, you must ask the TAs via Piazza to discuss it. Also, when you ask about using the external package, you need to demonstrate “Why do you have to use this package?” to TAs. If not asking via Piazza and using external packages, **we regard this behavior as cheating**.

# **Misc**

The only deadline extension allowed is through the use of tokens obtained by participating in the UniCode programming contest. (Details will be announced by the UniCode organizers.) No other extensions will be permitted.

Be aware of plagiarism! Although it is encouraged to discuss with others and refer to extra materials, copying other students or opened code is strictly banned. The TAs will compare your source code with other students' code. If you are caught, you will receive a severe penalty for plagiarism.

Last semester, we found a couple of plagiarism cases through an automated tool. Please do not try to cheat TAs. If you have any requests or questions regarding administrative issues (such as late submission due to an unfortunate accident, git push or repository not working) please ask TAs through Piazza.

Be careful not to post your own code when posting questions on Piazza. If you post your own code, it will be considered an act of plagiarism. And please keep your questions private if your questions can be a hint for other students to do their assignment.

If possible, it is better to ask questions through Piazza (assignment - #1), but if it is urgent, you can send an email to TAs (sieun908@unist.ac.kr / dmstjd517@unist.ac.kr).