Project 1. MIPS Assembler

Due 23:59, Sept. 26th

TA: Seonjin Na

1. Introduction

The objective of the first project is to implement a **MIPS ISA assembler**. The assembler is a tool which converts assembly codes to a binary file. This project is intended to help you understand the MIPS ISA instruction set.

The assembler you are going to design is a simplified assembler which does not support linking process, and thus you do **not need to add the symbol and relocation tables** for each.

You should implement the assembler which 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.

*****If you have any questions related to the project, please ask them on the Q&A board as possible as possible but you can ask them during office hours. You can ask TA about   high-level design issues or how to use systems or tool but not about debugging your code. The assigned TA(Seonjin Na) will answer them whenever possible.***

- Instruction Set

The detailed information regarding instructions are in the green card page of textbook

ADDIU	ADDU	AND	ANDI	BEQ	BNE	J
JAL	JR	LUI	LW	LA*	NOR	OR
ORI	SLTIU	SLTU	SLL	SRL	SW	SUBU

- Only instructions for unsigned operations need to be implemented. (addu, addiu, subu, sltiu, sltu, sll, srl)

- However, the immediate fields and offset fields for certain instructions are sign extended to allow negative numbers (addiu, sltiu, beq, bne, lw, sw). So you must implement the assembler to read the fields as signed-extended bits.

- Only loads and stores with 4B word 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 ranges 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

- Directives

.text
- indicates that following items are stored in the user text segment, typically instructions
- It always starts from <u>0x400000</u>

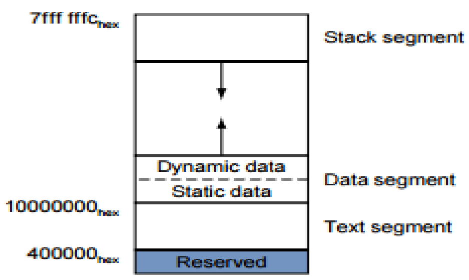
.data
- indicates that following data items are stored in the data segment
- It always starts from <u>0x10000000</u>

.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 (Each word has an initial value).

- Memory Layout

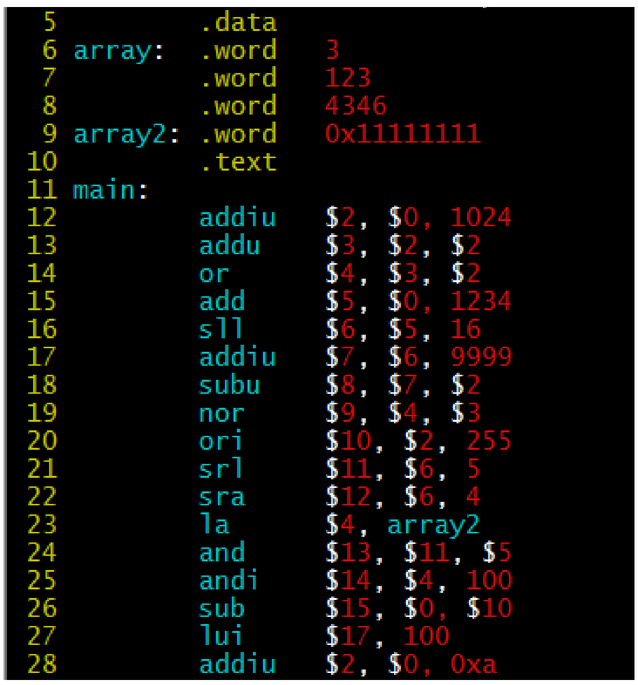


- Execution command:

> ./runfile <assembly file>

Your program must produce a single output file (.*o) from the input assembly file (*.s).

- Input format



```
        .data
array:  .word   3
        .word   123
        .word   4346
array2: .word   0x11111111
        .text
main:
        addiu   $2, $0, 1024
        addu    $3, $2, $2
        or      $4, $3, $2
        add     $5, $0, 1234
        sll     $6, $5, 16
        addiu   $7, $6, 9999
        subu    $8, $7, $2
        nor     $9, $4, $3
        ori     $10, $2, 255
        srl     $11, $6, 5
        sra     $12, $6, 4
        la      $4, array2
        and     $13, $11, $5
        andi    $14, $4, 100
        sub     $15, $0, $10
        lui     $17, 100
        addiu   $2, $0, 0xa
```

- Output format

The output of the assembler is an object file(*.o file) which contains a single string of **ASCII ‘0’ and ‘1’** characters. The ASCII string follows 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 format of binary ASCII string :

<text section size>
<data section size>
<instruction 1>
…
<instruction n>
<value 1>
…
<value m>

Please refer to the given examples for a better idea of this format.

2. Program Language

You can choose the programming language among C, and C++. Since subsequent project 2, 3, and 4 should be written in C/C++, you may want to start with C/C++ for the project to get familiar with the language, if you are not yet.

3. GitLab Repository

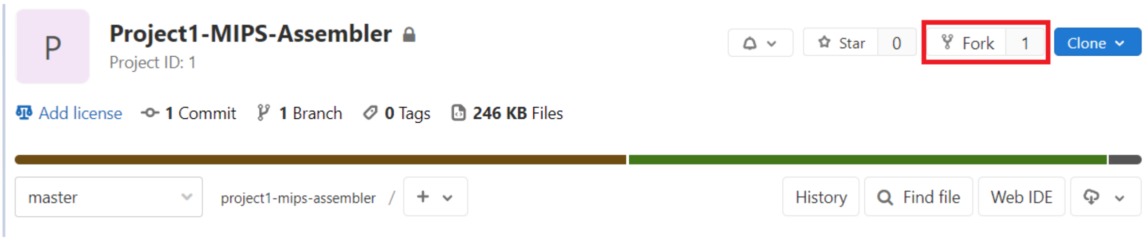
You must create a new repository for this project in GitLab. Please follow the instructions below step-by-step.

First, let's fork the project which the TAs have prepared.

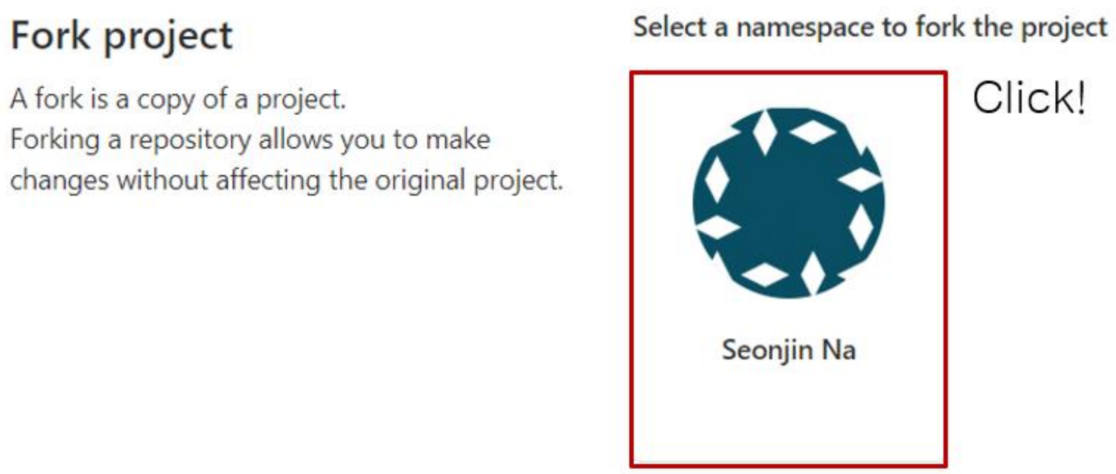
1) Access http://cs311.kaist.ac.kr:10080/cs311_projects/project1-mips-assembler
Examples can be found in the project.

2) (If prompted) Login with your GitLab account.

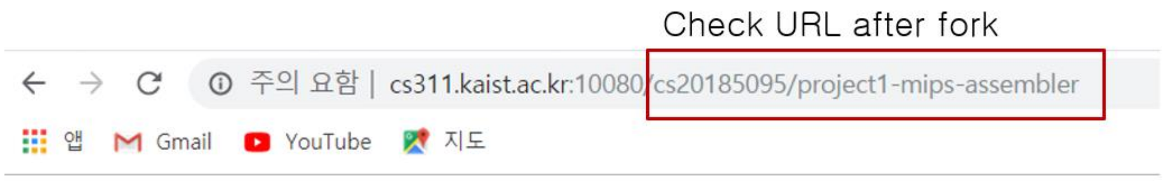
3) Click **"Fork"** (Please refer to the following screen shot)



4) **Choose** your personal account (Refer to the following screen shot)


Fork project
A fork is a copy of a project.
Forking a repository allows you to make changes without affecting the original project.
Select a namespace to fork the project
Click!
Seonjin Na

5) Wait for git to import repository and make sure your repository is **made for your group** by checking the URL at the top of your browser. There should be your cs+[your student id].

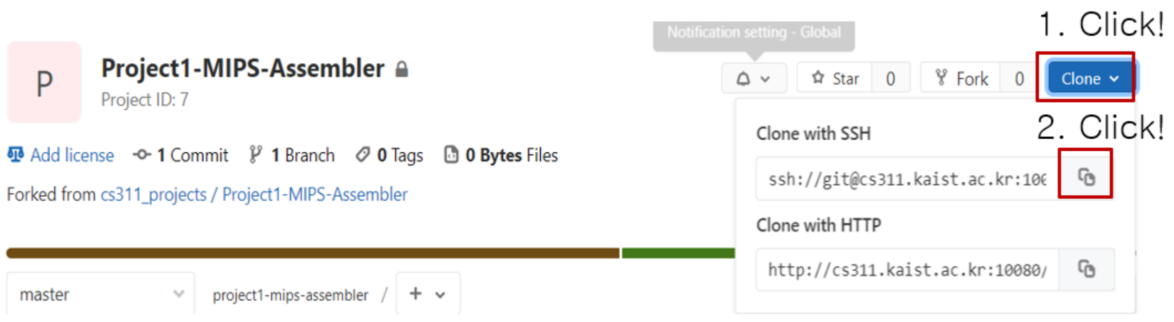

Check URL after fork
주의 요함 | cs311.kaist.ac.kr:10080/cs20185095/project1-mips-assembler
앱
Gmail
YouTube
지도

6) Congratulations! You are ready to create a local clone of your repository.
Next, let's make a local copy. In order to work on your project you must make a local copy first.

Before you proceed, the TAs would like you to take a look at the notice on KLMS which is related to best practices on how to set/use git for your own sake.

1) Before you can copy a repository from the server, you are required to register your public ssh key to the server. Please refer to the following for registration.
https://docs.gitlab.com/ce/ssh/README.html

2) After you have finished registration, take a look of the red boxed region and copy it to your clipboard (you can see the below figure). Your URL should look similar as the following:
ssh://git@cs311.kaist.ac.kr:10022/cs+your_student_id/Project1-MIPS-Assembler.git


Project1-MIPS-Assembler
Project ID: 7
Add license
1 Commit
1 Branch
0 Tags
0 Bytes Files
Forked from cs311_projects / Project1-MIPS-Assembler
Notification setting - Global
Star 0
Fork 0
1. Click!
Clone
2. Click!
Clone with SSH
ssh://git@cs311.kaist.ac.kr:10
Clone with HTTP
http://cs311.kaist.ac.kr:10080/
master
project1-mips-assembler /

3) Type the following in your working directory
> git clone YOUR_URL
Example) git clone ssh://git@cs311.kaist.ac.kr:10022/cs+your_student_id/project1-mips-assembler.git

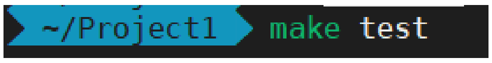
4) If you are successful at making a local copy, you will see a directory called “Project1-MIPS-Assembler”. Under this directory you will see the tar file for examples. Decompress the file by typing the following in that directory
> tar xvf input.tar
> tar xvf output.tar

Please remember that you are using the local repository as a working space. If you want to commit or submit your work, you must push your work to the remote server. Refer to the link below on how to push your work to the remote repository. (*Focus on **‘commit’** and **‘push’***)
Link: https://rogerdudler.github.io/git-guide/index.html

4. Grading Policy

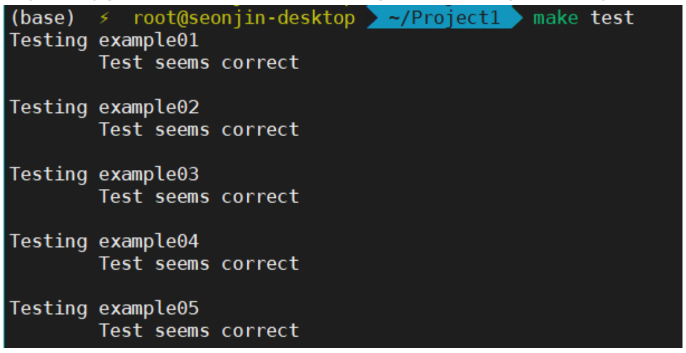
Grades will be given based on the 5 examples provided for this project. Your assembler should print the correct corresponding binary code for a given MIPS code.

If you want to check whether your assembler works correctly or not, you can type this command in project1 working directory.



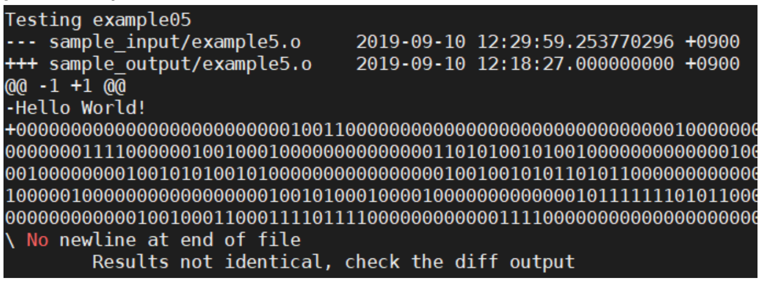
```
 ~/Project1  make test
```

If you type above command (make test), then you can see the results for each test case.



```
(base)  ⚡ root@seonjin-desktop  ~/Project1  make test
Testing example01
        Test seems correct

Testing example02
        Test seems correct

Testing example03
        Test seems correct

Testing example04
        Test seems correct

Testing example05
        Test seems correct
```

If your code’s output is different from answer, then the result shows the difference between your output and answer.



```
Testing example05
--- sample_input/example5.o     2019-09-10 12:29:59.253770296 +0900
+++ sample_output/example5.o    2019-09-10 12:18:27.000000000 +0900
@@ -1 +1 @@
-Hello World!
+0000000000000000000000000100110000000000000000000000000000001000000
00000001111000000100100010000000000000000110101001010010000000000000100
00100000000100101010010100000000000000000100100101011010110000000000000
10000010000000000000000010010100010000100000000000001011111110101100
0000000000001001000110001111011110000000000000001111000000000000000000
\ No newline at end of file
        Results not identical, check the diff output
```

There are 5 codes to be graded and you will be granted 20% of total score for each correct binary code and **being "Correct" means that every digit and location is the same** to the given output of the example. If a digit is not the same, you will receive **0 score** for that example.

5. Submission (Important!!)

Make sure your code works well on your allocated Linux server.
In fact, it is highly recommended to work on your allocated server throughout this class. Your project will be graded on the same environment as your allocated Linux server.

You must include a Makefile in your submission that builds your 'assembler' into the name 'runfile'. Also you should not modify Makefile. What you need to modify is main.c
We will be building your assembler using the `make` command(we will use "make test" to grade your score) . An example Makefile for c is provided in the project directory.

If you want to add commit messages, all you need to do is fill in the part after the option '–m' when committing your work.

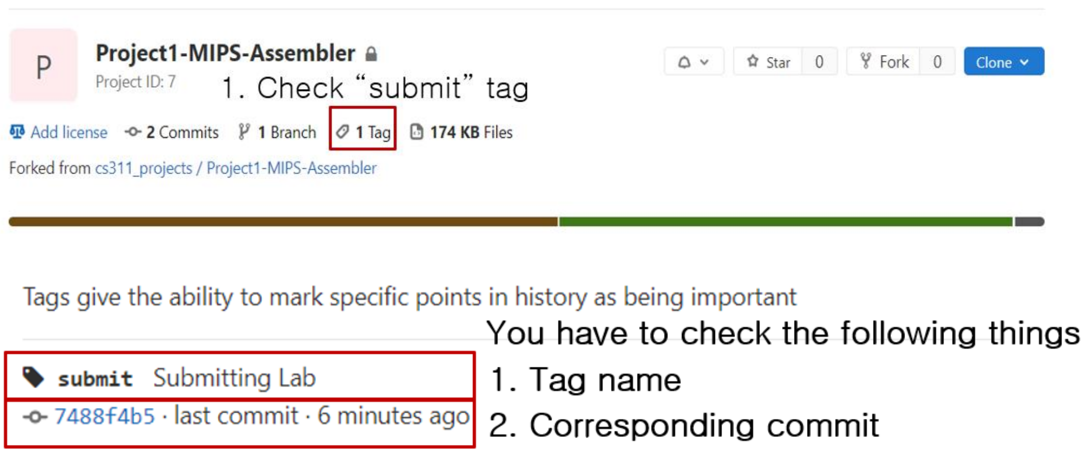
> git commit –m "TYPE_HERE_YOUR_MESSAGE"

Submit your work to your GitLab repository by adding a "submit" tag.
Please follow the steps below when submitting.

1) Commit and push your code, Makefile to your remote repository.

2) Type the following command in your working directory.
2-1) > **git tag –a submit –m '*whatever message you want*'**
2-2) > **git push origin submit**

After typing above commands, you should check your repository.



Forked from cs311_projects / Project1-MIPS-Assembler

Tags give the ability to mark specific points in history as being important

submit Submitting Lab
7488f4b5 · last commit · 6 minutes ago

You have to check the following things
1. Tag name
2. Corresponding commit

If there is no “submit” tag, your work will not be graded so please remember to submit your work with the tag.

6. Late Policy

You will loss **50%** of your score on the **first day** (Sept 27th 0:00 ~ Sept 27th 23:59). We will **not accept** works that are submitted after then.

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 be comparing your source code with open source codes and other team’s code. If you are caught, you will receive a penalty for plagiarism.

If you have any requests or questions regarding administrative issues (such as late submission due to an unfortunate accident, GitLab is not working) please send an e-mail to the TAs(cs311_ta@calab.kaist.ac.kr)