CS311 Project 2: Building a Simple MIPS Simulator

Due 11:59pm, October 31st, 2017
TA: Taekyung Heo

1. Overview

This second project is to build a simulator of a subset of the MIPS instruction set. The simulator loads a MIPS binary into a simulated memory, and executes the instructions. Instruction execution will change the states of registers and memory. Please also read the README.md file provided in the repository.

** If you have any questions related to the project, please ask them on the Q&A board. The assigned TA(Taekyung Heo) will answer them whenever possible.

2. Simulation Details

For a given input MIPS binary (the output binary file from the assembler built in Project 1), the simulator must implement the behaviors of the MIPS ISA execution.

2.1 States

The simulator must maintain the system states, which consist of the necessary register set (R0-R31, PC) and memory. The register and memory must be created when simulation begins.

2.2 Loading an input binary.

For a given input binary, the loader must identify the text and data section sizes. The text section must be loaded to the simulated memory from the address 0x400000. The data section must be loaded to the simulated memory from the address 0x10000000. In this project, the simple loader does not create the stack region.

2.3 Initial states

* PC: The initial value of PC is 0x400000.
* Registers: All values of register 0 to 31 are set to zero.
* Memory: You may assume all initial values are zero, except for the loaded text and data sections.

2.4 Instruction execution

With the current PC, 4 bytes from the memory is read. The simulator must parse the binary instruction and identify what the instruction is and what the operands are. Based on the MIPS ISA, the simulator must accurately implement the execution, which will update PC, register, or memory.

2.5 Completion.

The emulator must stop after executing the given number of instructions.

2.6 Supported Instruction Set

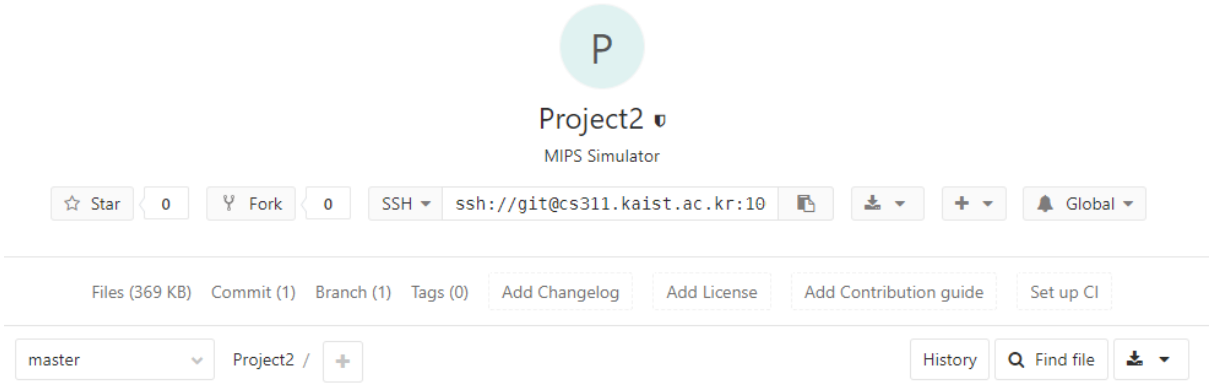
ADDIU	ADDU	AND	ANDI	BEQ	BNE	J
JAL	JR	LUI	LW	LA*	NOR	OR
ORI	SLTIU	SLTU	SLL	SRL	SW	SUBU

3. Forking and Cloning your Repository

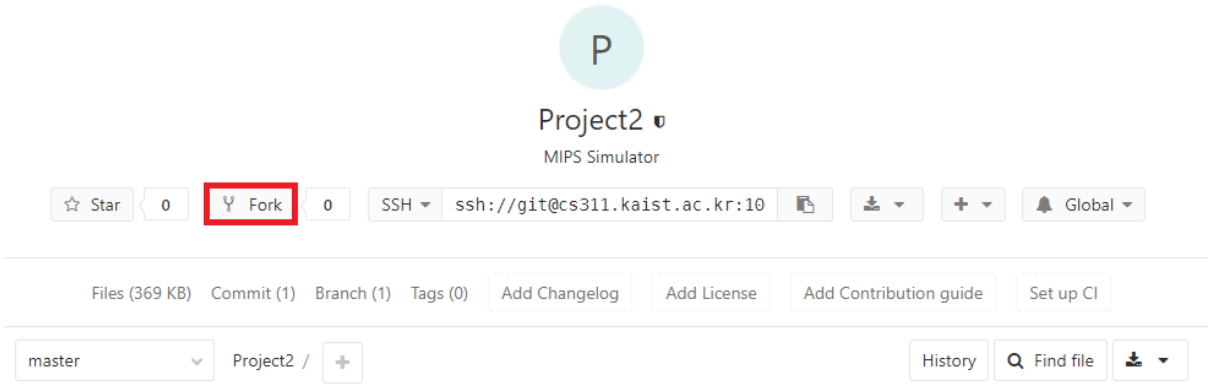
As with Project 1, we will fork the TA's Project2 repo to your team namespace. Then you will clone your team's repo into your local machines to work on the project.

3.1 Forking the TA's Repo

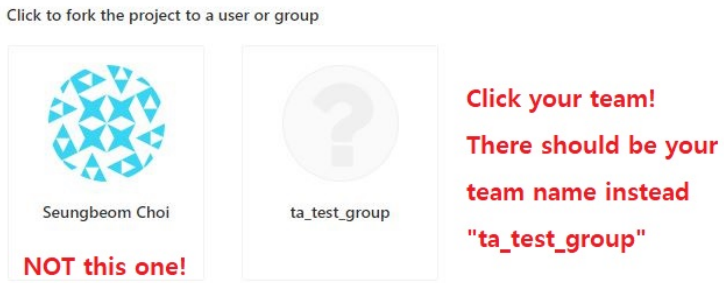
(1) Go to the following page: https://cs311.kaist.ac.kr/TAs/Project2. The page is the TA's epository.



(2) Click the fork button just like Project 1.



(3) Now select your Team (NOT YOUR USER) and the repo will be forked.



(4) Your Team repo will have the following URL: https://cs311.kaist.ac.kr/Team[your number]/Project2

3.2 Cloning the team repository to your local machine

From the website of your team repo copy the SSH or HTTPS URL of the git repository
the SSH URL will look something like the following:

```
ssh://git@cs311.kaist.ac.kr:10022/Team[your number]/Project2.git
```

Change directory to the location you want to clone your project and clone!

```
$ git clone ssh://git@cs311.kaist.ac.kr:10022/Team[your number]/Project2.git
```

Be sure to read the README.md file for some useful information.

4. Simulator Options and Output

4.1 Options

```
$ ./cs311sim [-m addr1:addr2] [-d] [-n num_instr] inputBinary
```

- -m : Dump the memory content from addr1 to addr2
- -d : Print the register file content for each instruction execution. Print memory content too if –m option is enabled.

The default output is the PC and register file content after the completion of the given number of instructions. If –m option is specified, the memory content from addr1 to addr2 must be printed too.

If –d option is set, the register (and memory dump, if –m is enabled) must be printed for every instruction execution.

4.2 Formatting Output

PC and register content must be printed in addition to the optional memory content.
You should print the output with standard output.

1. If you type the command line as below, the output file should show only PC and register values like Figure 1.

```
$ ./cs311sim -n 0 input.o
```

2. If you type the command line as below, the output file should show memory contents of specific memory region, PC and register values like Figure 2.

```
$ ./cs311sim -m 0x400000:0x400010 -n 0 input.o
```

3. The functions for printing the memory and register values are provided in the util.c, util.h file.

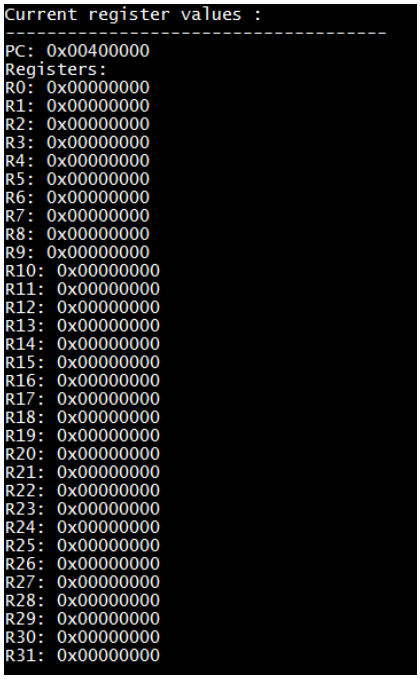


Figure 1. Dump Register Values

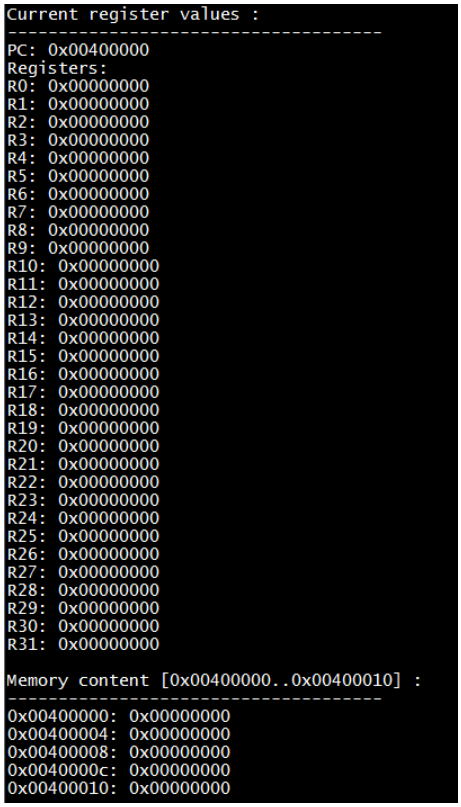


Figure 2. Additionally dump memory

5. Grading Policy

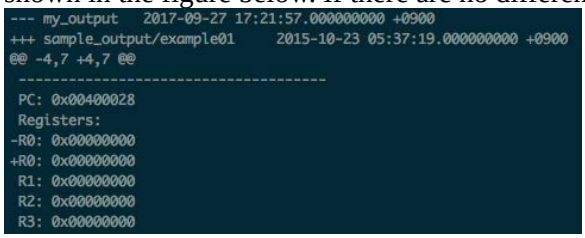
Grades will be given based on the 7 examples provided for this project provided in the `sample_input` directory. Your simulator should print the exactly same output as the files in the `sample_output` directory.

We will be automating the grading procedure by seeing if there are any difference between the files in the `sample_output` directory and the result of your simulator executions.
Please make sure that your outputs are identical to the files in the sample_output directory.

You are encouraged to use the `*diff*` command to compare your outputs to the provided outputs.

```
$ ./cs311sim -m 0x10000000:0x10000010 -n 50 sample_input/example01.o > my_output
$ diff -Naur my_output sample_output/example01
```

If there are any differences (including whitespaces) the diff program will print the different lines, as shown in the figure below. If there are no differences, nothing will be printed.



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 the example.

6. Submission (Important!!)

6.1 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.

6.2 Summarize the contribution of each team member.
If you are working with your teammate, you need to summarize your contributions to each project. If you use good commit messages, this can be done in a simple step. **"git shortlog"** summarizes commit titles by each user and will come in handy (especially if your commit titles have useful information).
If you want to add commit messages, please fill in the part after the option '–m' when committing.

```
$ git commit
```

A text editor will pop up with some information about the commit. Fill out your commit message at the top. The first line is the subject line of the commit. The second line should be blank, the third line and onwards will be the body of your commit message.

6.3 Add the 'submit' tag to your final commit and push your work to the gitlab server.
The following commands are the flow you should take to submit your work.

```
$ git tag submit
$ git push
$ git push --tags
```

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

If you do not `push` your work, we will not have the visibility to your work. Please make sure you push your work before the deadline.

7. Late Policy

You will lose **50%** of your score on the **first day** (Nov 1st 0:00~23:59). 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 compare 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).

8. Updates/Announcements

If there are any updates to the project, including additional tools/inputs/outputs, or changes, we will post a notice on the Notice board of KLMS, and will send you an e-mail using the KLMS system. Check your KLMS linked e-mail account for updates.