SCE212 Project 2: Building a Simple MIPS Simulator

Due 11:59pm, May 23th

1. Overview

This project is to build a simulator of a subset of the MIPS instruction set. The simulator loads a MIPS binary into an simulated memory, and execute the instructions. Instruction execution will change the states of registers and memory.

2. Simulation Details

For a given input MIPS binary (the output binary file from the assembler built in Project 1), the simulator must be able to mimic 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 the memory. The register and memory must be created when the 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 register0 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, 4B from the memory is read. The simulator must parse the binary instruction and identify what the instruction is and what are the operands. Based on the MIPS ISA, the simulator must accurately mimic the execution, which will update either a PC, register, or memory.

2.5 Completion

The simulator must stop after executing a give number of instructions.

2.6 Supported instruction set (same as Project 1)

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

Like Project 1, you will fork the SCE212/Project2 repo to your student ID namespace. Then you will clone your repo into your local machines to work on the project. Following instruction is identical to the Project1.

3.1 Forking the Class's repo

(1) Go to the following page: http://sce212.ajou.ac.kr/2019S-F039-2/project2. The page is your class repository.
(2) Click the fork button.
(3) Select your account and the repo will be forked.
(4) Your repo will have the following http://sce212.ajou.ac.kr /[your student ID]/project2
(5) NOTE: We will be running automated scripts to download your work and grade your projects. **Please do not change the name of your project paths**. (Keep the project name & path as `project2`)

3.2 Cloning your repository to your local machine (your virtual machine)

From the website of your repo, you can copy the HTTP URL of the git repository. The HTTP URL will look something like the following:

```
http://sce212.ajou.ac.kr/[your student ID]/project2.git
```

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

```
$ git clone http://sce212.ajou.ac.kr/[your student ID]/project2.git
```

The second command (HTTP) requires below entries when you type
Username for 'http://sce212.ajou.ac.kr': <**your student ID**>
Password for 'http://[your student ID]@sce212.ajou.ac.kr': <**your password**>

Be sure to read the `README.md` file for some useful information. It includes the explanation of each file and which files you are allowed to modify for this project.

4. Simulator Options and Output

4.1 Options

```
sce212sim [-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.
- `-n` : Number of instructions simulated

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`.

```
$ ./sce212sim -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`.

```
$] ./sce212sim -m 0x400000:0x400010 -n 0 input.o
```

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

```
Current register values :
--------------------------------------
PC: 0x00400000
Registers:
R0: 0x00000000
R1: 0x00000000
R2: 0x00000000
R3: 0x00000000
R4: 0x00000000
R5: 0x00000000
R6: 0x00000000
R7: 0x00000000
R8: 0x00000000
R9: 0x00000000
R10: 0x00000000
R11: 0x00000000
R12: 0x00000000
R13: 0x00000000
R14: 0x00000000
R15: 0x00000000
R16: 0x00000000
R17: 0x00000000
R18: 0x00000000
R19: 0x00000000
R20: 0x00000000
R21: 0x00000000
R22: 0x00000000
R23: 0x00000000
R24: 0x00000000
R25: 0x00000000
R26: 0x00000000
R27: 0x00000000
R28: 0x00000000
R29: 0x00000000
R30: 0x00000000
R31: 0x00000000
```

Figure 1. Dump Register Values

```
Current register values :
--------------------------------------
PC: 0x00400000
Registers:
R0: 0x00000000
R1: 0x00000000
R2: 0x00000000
R3: 0x00000000
R4: 0x00000000
R5: 0x00000000
R6: 0x00000000
R7: 0x00000000
R8: 0x00000000
R9: 0x00000000
R10: 0x00000000
R11: 0x00000000
R12: 0x00000000
R13: 0x00000000
R14: 0x00000000
R15: 0x00000000
R16: 0x00000000
R17: 0x00000000
R18: 0x00000000
R19: 0x00000000
R20: 0x00000000
R21: 0x00000000
R22: 0x00000000
R23: 0x00000000
R24: 0x00000000
R25: 0x00000000
R26: 0x00000000
R27: 0x00000000
R28: 0x00000000
R29: 0x00000000
R30: 0x00000000
R31: 0x00000000

Memory content [0x00400000..0x00400010] :
--------------------------------------
0x00400000: 0x00000000
0x00400004: 0x00000000
0x00400008: 0x00000000
0x0040000c: 0x00000000
0x00400010: 0x00000000
```

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 exact 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.

```
$ ./sce212sim -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. If there are no differences, nothing will be printed. Furthermore, we have provided a simple checking mechanism in the `Makefile`. Executing the following command will automate the checking procedure.

```
$ make test
```

There are 7 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

6.1 Make sure your code works well on the virtual machine environment we provided

In fact, it is highly recommended to work on the pre-created VM image from TA throughout this class. Your project will be graded on the same environment as the VM.

6.2 Summarize the contribution

You need to summarize your contributions to each project. Add a 'contribution.txt' file in your repository (don't forget to commit it). 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).

```
$ git shortlog > contribution.txt
$ git add contribution.txt
$ git commit
```

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

6.4 Updating Your Submit Tag

If you decide after tagging your commit and pushing, that you want to update your submission, you will need to remove the existing tag and retag & repush your submit tag.

```
$ git tag -d submit          # Deletes the existing tag
$ git push origin :submit    # Removes the 'submit' tag on the server
```

Now you may re-tag your work and submit using the instruction in Section 6.3

7. Updates/Announcements

If there are any updates to the project, including additional tools/inputs/outputs, or changes, we will post a notice on the Ajou BB, and will send you an e-mail using the Ajou BB system. **Frequently check your AjouBB linked e-mail account or the AjouBB notice board for updates.**