CSE261 Project 2: Building a Simple MIPS Simulator

Deadline : November 20

1. Overview

This 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.

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.

2.3 Initial States

- PC: The initial value of PC is 0x400000.
- Register: All values of *R0* to *R31* are set to zero, except *R29*, which is set to 0x80000000
- 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 Byte 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 the PC, registers, or memory.

2.5 Completion

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

2.6 Supported 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*	

3. About project 2

3.1 Download

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

1. Go to the each gitlab page. The page, which is same as assignment 1, will contain project 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.

Note that :

- `util.py` contains many useful functions, so, refer the comments.
- You must complete the skeleton code in `run.py` and `parse.py` to get score.
- You can modify `initialize.py`, `util.py`, `main.py`, if you want, but it is not mandatory.
- If you do not want to use the skeleton code, it is allowed to write code from scratch. However, you are supposed to follow the input and output file format.

4. Simulator Options and Output

4.1 Options

```
python3 main.py [–m] [–d] [–n num_instr ] inputBinary
```

- -m: Dump the written memory content
- -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

   ```
   python3 main.py –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.

   ```
   python3 main.py –m –n 0 input.o
   ```

3. The functions for printing the memory and register values are provided in the `util.py` file.

5. Grading Policy

You can test your assembler by using examples in the sample_input directory. Your assembler should print exactly 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 white spaces) 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>`)

2. Deduct 1 point for each different line.
3. If there are more than 10 different lines, 0 points are given.

6. 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.

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 BB and Piazza. Please check the notice for any updates. One concern is that we are using Python as an assignment language, there is plenty of packages you can leverage. Therefore, **we restrict you not to import external packages.** If you have a question about using an external package, you must ask 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**.

8. Misc

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 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 cheating. 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 - #2), but if it is urgent, you can send an email to TAs (dongu6213@unist.ac.kr / aero5010@unist.ac.kr / shjue@unist.ac.kr).