SCE212 Project 1: Implementing a MIPS Assembler

Due 11:50PM, April 24th

1. Overview

This project is to implement a MIPS (subset) ISA assembler. The assembler is the tool which 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.

The assembler is a simplified assembler which do 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 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.

2. Instruction Set

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

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 for certain instructions are sign extended to allow negative numbers (`addui`, `beq`, `bne`, `lw`, `sw`, `sltui`)

- 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 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
```

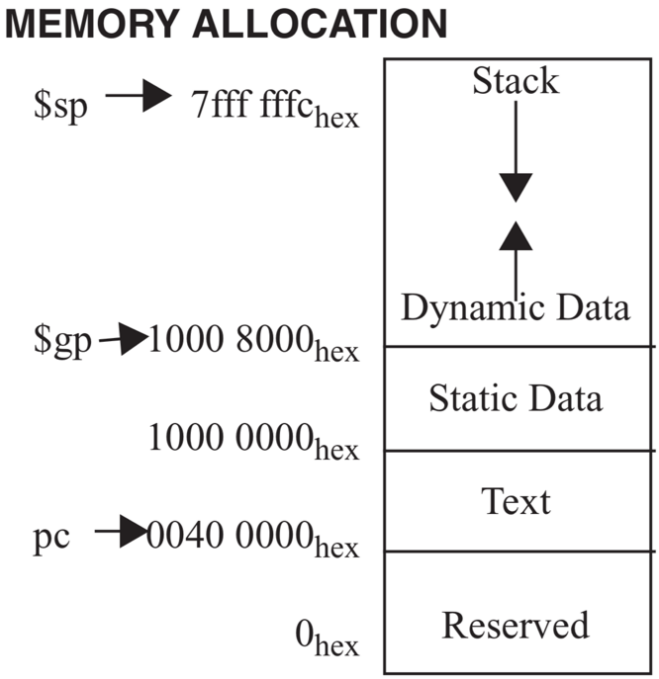
2.1 Directives

`.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 (Each word has an initial value). In the following figure, we illustrate the memory map used in our projects.



2.2 Input format

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

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>
```

3. Forking and Cloning your Repository

You will fork the SCE212/Project1 repo to your student ID namespace. Then you will clone your repo into your local machines to work on the project.

3.1 Forking the Class's repo

(1) Go to the following page: http://sce212.ajou.ac.kr/2019S-F039-2/project1. 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]/project1
(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 `project1`)

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

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

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

```
http://sce212.ajou.ac.kr/[your student ID]/project1.git
```

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

```
$ git clone http://sce212.ajou.ac.kr/[your student ID]/project1.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**>

If you typed correctly, you will get the clone repo in your virtual machine.

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. Grading Policy

Grades will be given based on the 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. 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 5 code segments 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.

5. Submission

5.1 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.

5.2 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.

5.3 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

5.4 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 5.3

6 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.**

7 Misc

We will accept your late submissions, but your score will lose up to 50%. Please do not give up the project.

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.

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, GitLab is not working) please send an e-mail to the TAs (heysid@ajou.ac.kr / tome01@ajou.ac.kr / limkim4233@ajou.ac.kr).