CSE26101 Project 1: Implementing a MIPS Assembler

Due 11:59 PM, Oct 9th

1. Overview

This project aims to help you understand the MIPS instruction set architecture (ISA) and be familiar with the principles of assemblers. You will implement a subset of the MIPS ISA assembler.

An assembler is a type of computer program that translates programs written in assembly code into machine code (binary). This output machine code is usually further processed by a linker to generate a single executable file. In this project, we focus on the assembler only, so you do not need to add the symbols and relocation tables for each file.

2. Instruction Set

Your assembler must

1) support the subset of the MIPS instructions listed in the following table. Detailed information regarding the instructions can be found on the attached MIPS green sheet page.
2) handle labels for jump/branch targets, and labels for the static data section.
3) support `push` and `pop` instructions that are not part of the MIPS ISA.

ADD	ADDI	ADDIU	ADDU	AND	ANDI	BEQ	BNE	J	JAL
JR	LUI	LW	LA*	NOR	OR	ORI	SLT	SLTI	SLTIU
SLTU	SLL	SRL	SW	SUB	SUBU	MOVE*	BLT*	PUSH*	POP*

* Pseudoinstructions

- For `lw` and `sw` instructions, the assembler only needs to support 4B words.

- The assembler must support decimal and hexadecimal numbers (0x) for the immediate field and `.data` section.

- The register name is always in "`$n`", where n is from 0 to 31.

- Pseudoinstructions:

    a) `la` (load address) should be converted to `lui` and `ori` instructions.

    `la $2, VAR1` → `lui $register`, upper 16bit address
    `ori $register`, lower 16bit address

    If the lower 16-bit address is 0x0000, the `ori` instruction should be omitted.

    Case1) load address is 0x1000 0000

    ```
    lui $2, 0x1000
    ```

    Case2) load address is 0x1000 0004

    ```
    lui $2, 0x1000
    ori $2, $2, 0x0004
    ```

b) `move` should be converted to a specific `add` instruction.

```
move $8, $9        →    addi $8, $9, 0
```

c) `blt` should be converted to `slt` and `bne` instructions.

```
blt $8, $9, L      →    slt $1, $8, $9
                        bne $1, 0, L
```

d) `push` is not a standard MIPS instruction. `push` should take a register as an operand and push the value of the register onto the top of the stack. `push` should be converted to `addi` and `sw` instructions.

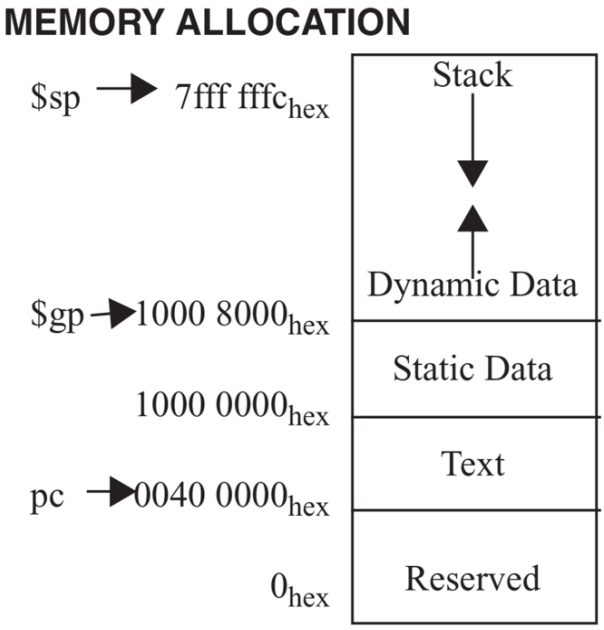
```
push $8            →    addi $29, $29, -4      * $29 is the stack pointer
                        sw $8 0($29)
```

e) `pop` is not a standard MIPS instruction. `pop` should take a register as an operand, pop the value on top of the stack, and store it in the register. `pop` should be converted to `lw` and `addi` instructions.

```
pop $8             →    lw $8 0($29)           * $29 is the stack pointer
                        addi $29, $29, 4
```

2.1 Directives

The following figure illustrates the memory map used in our projects.



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

`.data`
- indicates that the following data items are stored in the data segment.
- It always starts from `0x10000000`.

`.word`
- 4-byte (32 bits) consecutive memory.

You can assume that the `.data` and `.text` directives appear only once, and the `.data` must appear before the `.text` directive. Assume that each word in the data section is initialized (Each word has an initial value).

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 example of the input files. 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 (32-bit) are the size of the 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 the 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>
<initial values of the data section>
```

3. Testing and Submission via PA Submit System (PASS)

We will use ***PA Submit System***, developed by Systems Software Lab of Ajou University, to manage the programming assignments. Please refer to the BlackBoard announcement for the PASS user manual.

3.1 Downloading the Skeleton Code

The skeleton code is available on BlackBoard under "Assignment" → "PA1: Implementing a MIPS Assembler". Please download and read the skeleton code carefully before you begin working on PA1. Some macros and helper functions are given, please do not modify them. Please put your implementation under the "Fill the blanks" regions.

We will provide a few basic test cases along with the skeleton code. These test cases test a single functionality of your implementation (e.g., whether *addi* is implemented correctly). You can test on them in your local environments before submitting to PASS.

3.2 Submission

You should submit your assignment as a single Python file (assembler.py) on PASS, under "PA1: Implementing a MIPS Assembler". Please do not modify the file name or include unnecessary files in your submission.

Your code will be automatically tested with the test cases we uploaded. The test cases are based on combinations of MIPS instructions, each representing a complete, runnable code snippet.

You are allowed up to 100 submissions, **you can choose which submission will be used for grading** (please refer to the PASS User Manual).

4. Grading Criteria

Your assignment will be graded automatically on PASS. Note that the grading is only based on the test results on PASS. If your code fails to run on PASS (not your local environment), you will not get any points. **Your score on PASS is your final score of PA1.**

Please read the outputs of the test cases on PASS. In case of a failed test, it will provide hints on where you should improve. For instance, "*addi* is not correctly converted" means your assembler did not return the correct results for an *addi* instruction.

A test will stop and return an error message upon the **earliest** error.

For late submissions, your slip days will be automatically subtracted; post-slip-day submissions will receive a 20% penalty from your score for each day late.

5. Updates/Announcements

We will post a notice on BlackBoard and Pizza if there are any updates to the assignment. Please check BlackBoard and Piazza regularly.

Note that you are allowed to import and use external packages for this assignment. If you think it is necessary to use an external package, you must first get permission from the TAs by posting in Piazza. Your post must explain clearly why the package you intend to use is necessary. **Using an external package without permission will be regarded as plagiarism.**

7. Misc

Be careful about plagiarism! Last semester, we found a couple of plagiarism cases through an automated tool. If you are caught in "deep collaboration" with other students, you will split the score equally with your collaborators.

Please do not upload your code when posting in Piazza as it is considered cheating. If you think your question contains hints to other students, please make sure you post it as a "private post".
If you have any requests or questions (technical difficulties, late submission due to inevitable circumstances, etc.), please ask the TAs on Piazza.

We generally encourage the use of Piazza for discussions. However, for urgent issues, you can send an email to the TAs (minseok1335@unist.ac.kr(Head TA) / dyryu@unist.ac.kr / xinyuema@unist.ac.kr / garvel@unist.ac.kr).