Lab1. MIPS assembler

Due 4:00PM, Oct. 08

TA: Taehoon Kim, Seunghun Jeon

Introduction

The objective of the first project is to implement a **MIPS ISA assembler**. The assembler is the tool which converts assembly codes to a binary file. This project is intended to help you understand the MIPS ISA instruction set.

The assembler you are going to design 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 fil 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.

Instruction Set

The detailed information regarding instructions are in the green card page of textbook

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

Directives

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

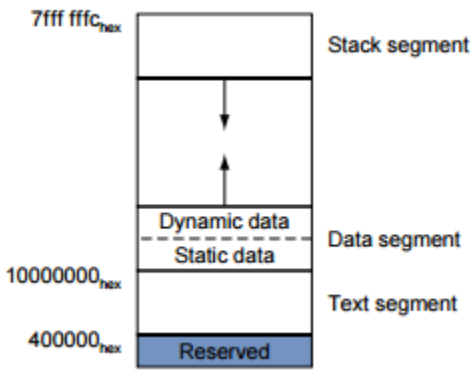
.data
- indicates that following data items are stored in the data segment
- It always starts from 0x10000000

.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).

Memory Layout



Execution command:

> ./runfile <assembly file>

Your program must produce a single output file (.*o) from the input assembly file (*.s).

Input format

```
 5          .data
 6 array:   .word   3
 7          .word   123
 8          .word   4346
 9 array2:  .word   0x11111111
10          .text
11 main:
12          addiu   $2, $0, 1024
13          addu    $3, $2, $2
14          or      $4, $3, $2
15          add     $5, $0, 1234
16          sll     $6, $5, 16
17          addiu   $7, $6, 9999
18          subu    $8, $7, $2
19          nor     $9, $4, $3
20          ori     $10, $2, 255
21          srl     $11, $6, 5
22          sra     $12, $6, 4
23          la      $4, array2
24          and     $13, $11, $5
25          andi    $14, $4, 100
26          sub     $15, $0, $10
27          lui     $17, 100
28          addiu   $2, $0, 0xa
```

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>

<value 1>
…
<value m>

Program Language

You can choose the programming language among C, C++, and Python. Since subsequent project 2, 3, and 4 should be written in C/C++, you may want to start with C/C++ for the project to get familiar with the language, if you are not yet.

Hand in

You should submit your code with your team name. (team_name.c/cpp/py)
If your team is team12, you should submit the code with team12.c or team12.cpp or team12.py.
Please send the email to cs311_ta@calab.kaist.ac.kr with attaching the code file until the due date.