CSC3050 Project 1 MIPS Assembler & Simulator

- Yongjin, Huang
- 119010115

Introduction

Implementation

String instead of integer in whole design of Assembler and Simulator

Scanner `Assembler::Scanner`

- `Assembler::Scanner::remove_comments` Remove comments, empty lines and tabs `\t`
- `Assembler::Scanner::split_data_and_text` Split data segment `.data` and text segment `.text` and it can handle multiple occurrences of `.text` and `.data`
- `Assembler::Scanner::preprocess_text`: preprocess data segment
  - Put label and its corresponding code together in the same line
  - Replace `,` with space
  - Throw tabs `\t`
- `Assembler::Scanner::scan`

Parser `Assembler::Parser`

- `Assembler::Parser::process_dataseg` Interpret data segment to machine code:
  - Based on different data types perform different operations:
    - `asciiz` `Assembler::Parser::get_ascii_data` Handle special characters like `\n` , `\t` , `\'` , `\"` , `\\`. Append `\0` terminator to its end.
    - `ascii` The same as `asciiz` except for `\0` terminator.
    - `word`
    - `half`
    - `byte`
  - Append zero or truncate to generate fixed length (32bits) machine code
- `Assembler::Parser::find_label` Locate all labels in text segment and store them using hashmap `unordered_map<string, uint32_t> label_to_addr` which maps label to address of the code.
- `Assembler::Parser::parse` Main function of `Assembler::Parser`:
  - Interpret each line of code in data segment into machine code

  - Based on different types of instruction, certain interpretation methods are performed:
    * R instructions
    * I instructions
    * J instructions
    * O instructions (`syscall` only)
  - And in each type of instructions, instructions of same format are grouped together
    * e.g. `sll` , `srl` , `sra` are in `op rd rt shamt` so they are grouped together
  - String concat to generate machine code
-

Simulator `Simulator`

- Memory structure:
  -
```
stack_st_idx = 6 * 1024 * 1024 = 6MB
| <- stack data
stack_end_idx
|
dynamic_end_idx
| <- dynamic data
dynamic_st_idx = static_end_idx
| <- static data
static_st_idx = 1 * 1024 * 1024 = 1MB
|
text_end_idx
| <- text data
text_st_idx = 0
```
  - 2D character array to simulate memory `std::array<std::array<char,8>,memory_size>`
  - integer array to simulate register `int32_t reg[reg_size]`
- Methods:
  - Retrieve or store data in memory:
    * `Simulator::get_word_from_memory`
    * `Simulator::get_byte_from_memory`
    * `Simulator::store_word_to_memory`
    * `Simulator::store_byte_to_memory`
  - Mapping between memory array index and address
    * `Simulator::addr2idx`
    * `Simulator::idx2addr`

Features & Tricks

Conversion between binary string `std::string` and integer `int32_t`

- int to string: `bitset<width>(int32_t).to_string()`
- string to int: `stoi(string,nullptr,base=2)`

Conclusion