CSC3050 Project 2

March 21, 2022

Build a MIPS simulator

Student ID: 120090266

Student Name: Feng Yutong

This assignment represents my own work in accordance with University regulations.

Signature: Feng Yutong

1 Overview

The task of this project is to implement a MIPS half-assembler and simulator. There are several challenges of this task:

1. Store data segment in proper byte order and in a way to load or store byte, half, word easily.

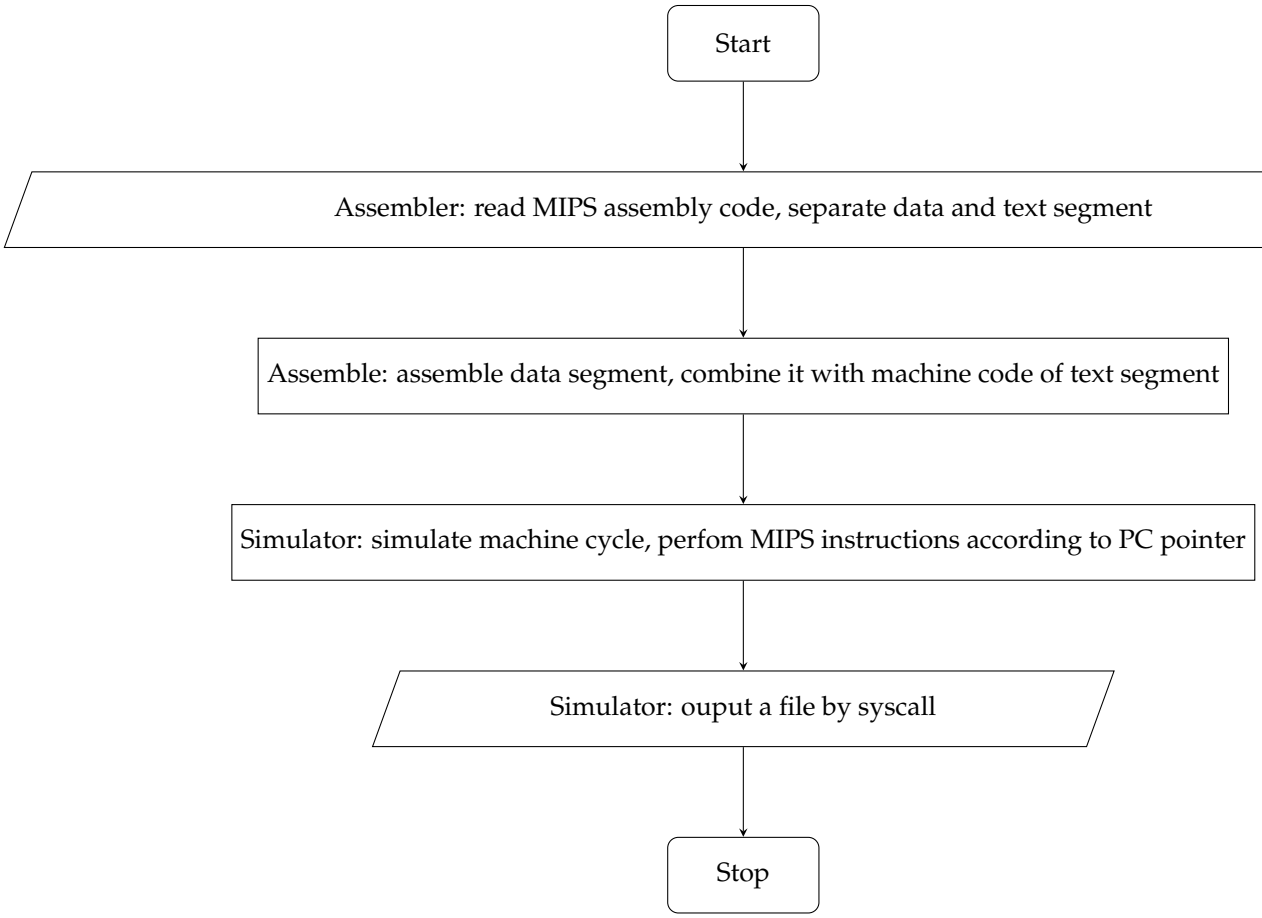
2. Design proper structure for memory, register and PC pointer.

3. Call MIPS functions according to machine code. Design function to imitate MIPS functions.

4. Design function to imitate MIPS syscall, which invokes the Linux APIs.

5. Imitate machine cycle and manage the change of PC pointer after each execution.

Since simulator should only simulate but not assemble, the code designs two class: Assembler and Simulator. The big picture thoughts and ideas are as followed:



2 High Level Implementation Ideas

There are two classes: Assembler and Simulator.

Assembler first assembles and stores data segment of MIPS assembly code. Then it combines data with machine code of text segment.

Simulator uses the machine code generating from Assembler and simulate.

2.1 Assembler

```
class Assembler{
        public:
        inline static vector<string> comb;
        void prepocess(istream &in);
        void combine(istream &in);
        void assemble(istream &in1, istream &in2);
        Assembler(){};
};
```

Assembler performs **assemble** function, which includes two process: **preprocess** and **combine**. Function **preprocess** separate data and text segment. Translate different data type into binary strings. Function **combine** slice and add place holder to make those binary numbers 32-bit. They are then stored with machine code for Simulator to use later.

2.2 Simulator

```
class Simulator{
        public:
        // files to use
        const vector<string> &input; // machine code
        istream &simin; // .in file
        ostream &simout; // .out file
        char* simcheck;
        // pointers
        static const int stk_ed = memory_tot, static_st = 1024*1024;
        int dyn_ed, text_ed = 0, static_ed = static_st;
        inline static char memory[memory_tot][8];
        int32_t pc;
        // registers and adrress
        int32_t &get_reg(const string &reg_str);
        int get_idx(int32_t addr);
        int32_t get_addr(int idx);
        // store and load
        void store(string unit, int32_t addr, string type);
```

```
19          string load(int32_t addr, string type);
20          int32_t load_word_val(int32_t addr);
21          int16_t load_half_val(int32_t addr);
22          int8_t load_byte_val(int32_t addr);
23          // main
24          void simulate();
25          void init();
26          void excuate(const string &instr);
27
28  };
```

Part of Simulator class are shown above. The class includes files, pointers, functions to store or load between registers and memory, convert between real and simulated adrress. Simulator performs **simulate** functions, which first calls **init** funtion. In initiation, program sotre static data and text, assign initial values to pointers. The memory structure are represented by those pointers as below:

```
1           | <- stk_ed = 6 * 1024 * 1024 = 6MB
2   stack   |
3           | <- dyn_ed
4   dynamic |
5           | <- static_ed
6   static  |
7           | <- static st = 1024 * 1024 = 1MB
8           | <- text_ed
9   text    |
10          | <- text_st = 0
```

Then it simulate machine cycle and **excuate** each instructions.

```
1   while(pc >= mem_base && pc < get_addr(text_ed)){
2           string word_str = load(pc, "word");
3           excuate(word_str);
4           pc += 4;
5   }
```

3 Implement Details

3.1 Data structure

Utilize various data type in C++ to make code neat and easy.

Use fixed width intger types: int32_t for word, int16_t for half, int8_t for byte, uint_32t for memory.

Use unit32_t etc. to solve MIPS instructions based on unsigned number.

Utilize logic operations to make sign extent easily and implement lwl, lwr, swl, swr. For example:

```
if num >> (num.len - 1) == 1: // sodu code
        num |= ((1<<(32-num.len)-1) << num.len) // 32-bit extent
```

Utilize STL containers to make conversion easier.

Use **map <string, int> reg_idx** and **int32_t reg[]** to get and store register.

Use **bitset<fixed width>(32-bit integer).to_string()** to convert between binary number and string.

Use **vector<string> comb** to restore the 32-bit-long machine code in string format. Since there exits operations of slicing and adding place holder, insert method helps change the order of the content in vector dynamically.

Use 2D array **char memory[memory_tot][8]** for memory.

Utilize inline static of c++17 to make code more neat.

3.2 Method

Use **store(string unit, int32_t addr, string type); string load(int32_t addr, string type); int32_t load_word_val(int32_t addr); int16_t load_half_val(int32_t addr); int8_t load_byte_val(int32_t addr)** to load and store value between registers and memory.

Use **int32_t &get_reg(const string &reg_str)** to get and modify the value stored in registers. For example:

```
void MIPS_add(const string &rd, const string &rs, const string &rt){
        get_reg(rd) = get_reg(rs) + get_reg(rt);
}
```

Utilize Linux APIs to implement syscall, such as open, read and write file.

Other global methods:

dec2bin: convert a integer (in string format) to its binary expression (in string format with fixed width).

ascii2bin: convert a string to its binary expression (in string format).

im2int: convert immediate (in string format) from binary to decimal.

extend: sign extend a decimal number.

4 Compile and run

4.1 Complie and run

The program requires environment of c++17. To run the program, move to the folder of main.cpp and enter following instructions in terminal:

```
$ make
$ ./simulator {MIPS file} {machine code file} {checkpts file}
{input file} {output file}
//e.g. ./simulator fib.asm fib.txt fib_checkpts.txt fib.in fib.out
```

The program dumps files at checkpoints for checking registers and memory.