Program 2 report

陈启旭 120090643

1. Big picture thoughts and ideas

   A MIPS simulator is a program that simulates the execution of a binary file.

   There are several inputs for the program:

   A MIPS file that contains MIPS assembly language code to roll the static data information.

   A Text file to enroll all the binary codes into the simulator

   A checkpoint file to help in debugging

   There are two primary processes for simulating the binary file:

   A. The first part is to put everything in the right place. The program will put the static data and the binary codes in the right place.
   B. Simulating: the program will get the corresponding instructions stored in the memory according to PC. Every time after fetching the instructions, the program will update the value of the PC. The program will execute the next instructions according to the updated PC.

2. High level implementation ideas
   a) Put everything in the right place:
      A. Overview:

         The total memory block is 6MB. The program uses a list which contains 6*2^20 elements to represent 6 MB. Each element contains 8 bits or 1 byte.

         The first address of the first element of the list is noted as 0x400000hex, which is 4194304 in decimal form.

         From 0x400000 to 0x500000, the list will store all the instructions in the binary file.

         The Static data part starts from 0x500000(5242880 in decimal form). The program will put the data in the .data segment of the MIPS file piece by piece in the static data part.

         At the place where the static segments end, the dynamic data part will start.

         The stack segment starts from the top of the 6MB.
      B. Enroll the text segment.

         The program will read through the txt file line by line. For every line of the txt file, the program will split it into 4 parts with 8 bits in each part. The 6MB list will append the all the binary codes in sequential (each individual part as an element)
      C. Enroll the static data

         See the implementation detail
   b) Simulating
      A. To get a line of machine code stored at the address PC specifies, go to the simulated memory.
      B. Update the PC
      C. Execute the code
   c) Executing
      A. The program will distinguish which instructions it is according to the opcode and funct

         If the code is 00000000000000000000000000001100, then the code is a syscall instruction.

If the opcode is “000000”, then the operation is R type. The program will split the 32 bits code into 6|5|5|5|5|6

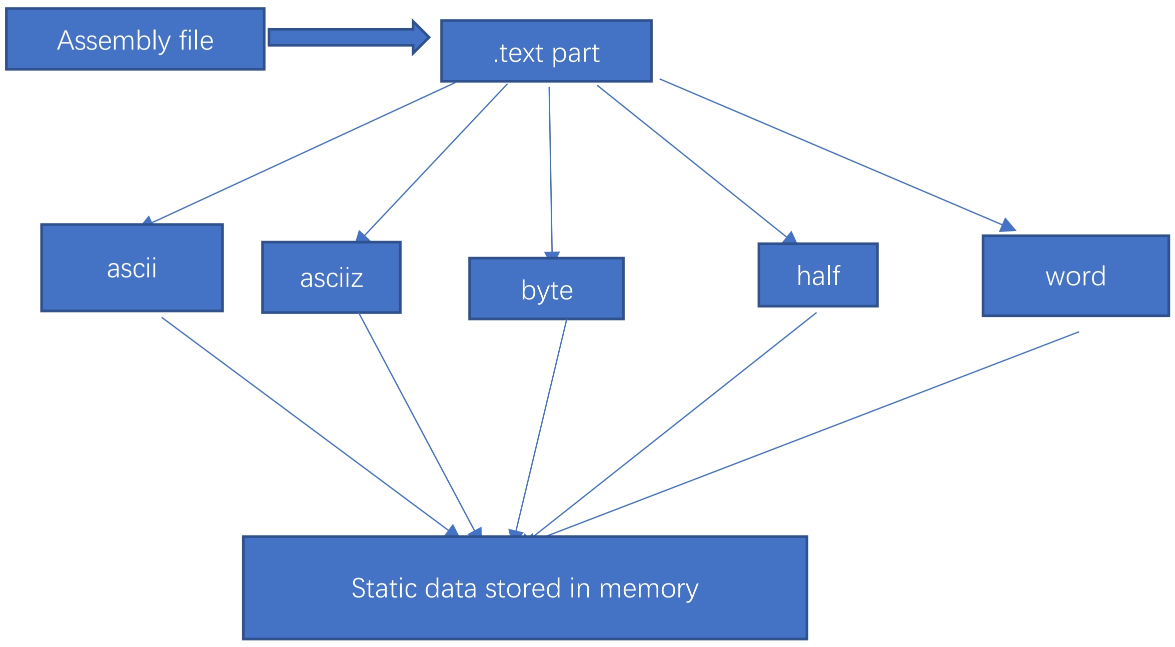
If the opcode is 000010 or 000011, then the operation will be I type. The program will split the 32 bits into 6|5|5|16

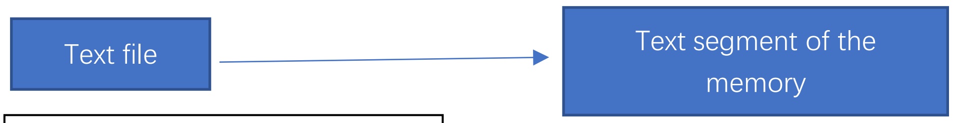
Else, the operation code is J type, the program will split the 32 bits into 6|26

B. The program will execute the instructions according to defined functions

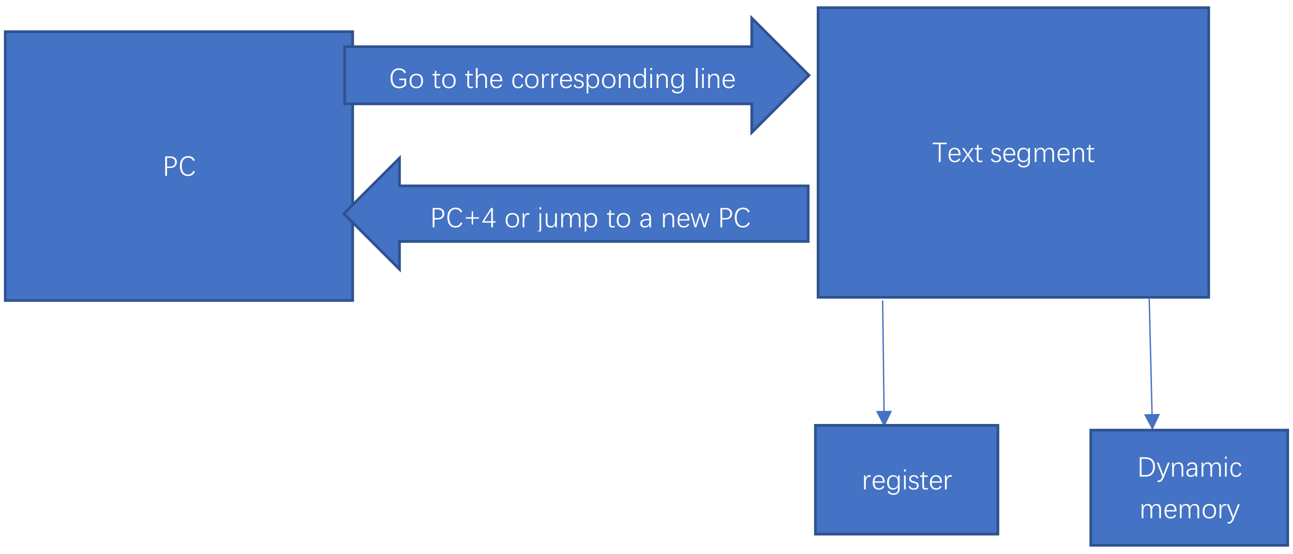
d) Workflow for the whole process

1. Stimulate the memory





2. execute the instructions



3. The implementation details

3.1 Data structure

Register_list: a python list that stores all the value of the registers. The first element stores the value of $zero, the second element stores the value of $at…the 32th element stores the value of $ra.

Hi_Lo: a list storing the value stored in Hi and Lo registers

Pc: a list storing the program counter

Function dictionary: it is 13 dictionaries which stores the opcodes and funct codes and their corresponding function

3.2 Some special detail

3.2.1 How to enroll the static data

i. The program will use the phase1 function, which is the same as the phase1.py in the first programming project. The function will create a map to store the string and its corresponding data type.
(all the data are stored in the form of little endian except ascii and asciiz)

ii. The memory space from 0x400000 to 0x500000 will create a space for the data according the data type. The program will store the data in the created space

3.2.2 How to execute the instructions

The program have contained functions for instructions to do what they are supposed to do

For example, for the add operation:

```
def R_add(rs,rt,rd):
    binary1 = extend2word(lift_value(rs))
    binary2 = extend2word(lift_value(rt))
    result = add(binary1,binary2)
    Register_list[int(unbinary2int(rd))]=extend2word(result)
```

The extend2word function will transverse a signed binary to a 32 bits code

The add function is special function which can operate two binary codes adding. For example: 1010 add 0001 will return 1011

After distinguishing the instructions type of the instructions, the program will read the opcodes or the funct codes of the instructions.The program will then invoke the dictionary like the following example to execute the instruction.

```
elif line[0:6]=="000000":
        rs = line[6:11]
        rt = line[11:16]
        rd = line[16:21]
        shamt = line[21:26]
        funct = line[26:]
        if funct in R_noshift_funct_dic.keys():
            R_noshift_funct_dic[funct](rs,rt,rd)
```

```

R_noshift_funct_dic={"100000":R_add,"100001":R_addu,"100010":R_sub,
"100011":R_subu,"100100":R_and,"100101":R_or,"100110":R_xor,
"100111":R_nor,"101010":R_slt,"101011":R_sltu,"000100":R_sllv,"000110":
R_srlv,"000111":R_srav}
```

3.2.3 Memory dump

4. How to test the code

   Input the following command like:

   python simulator.py memcpy-hello-world.asm memcpy-hello-world.txt memcpy-hello-world_checkpts.txt memcpy-hello-world.in memcpy-hello-world.out

   The user then can see the result in the .out file

   If the output is not the same as the expected output, please check the memory.bin and the register.bin file.