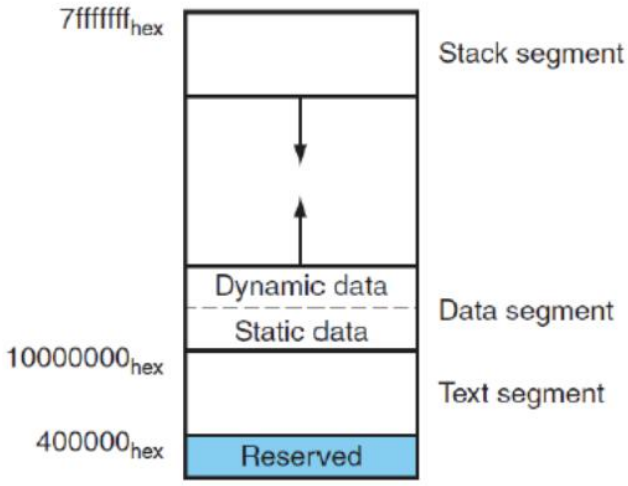
CSC3050 Project2 Report

Liang Mingrui 120090723

2022.3.19

Big Picture Thoughts

MIPS assemblers firstly scan the whole asm file, and then store the data in the .data part to the static data. In this project, the address of static data starts at 0x500000. Then the assembler will store the machine code that were changed from .text part to the text segment. And the address of it starts at 0x400000 in this project. The rest parts of the places is used to store the dynamic data and stack segment. In this project, there are totally 6mb to store all of this. Like the following picture:



Usually, the content are aligned, 4byte as a part.

In the whole process, the pc will begin at the 0x400000. Assemblers will excute machine code stored in the 0x400000. After that, the pc will point to the place accroding to the last instruction or directly add four. Then the assemblers will excute next machine code. This will not stop until the exit syscall.

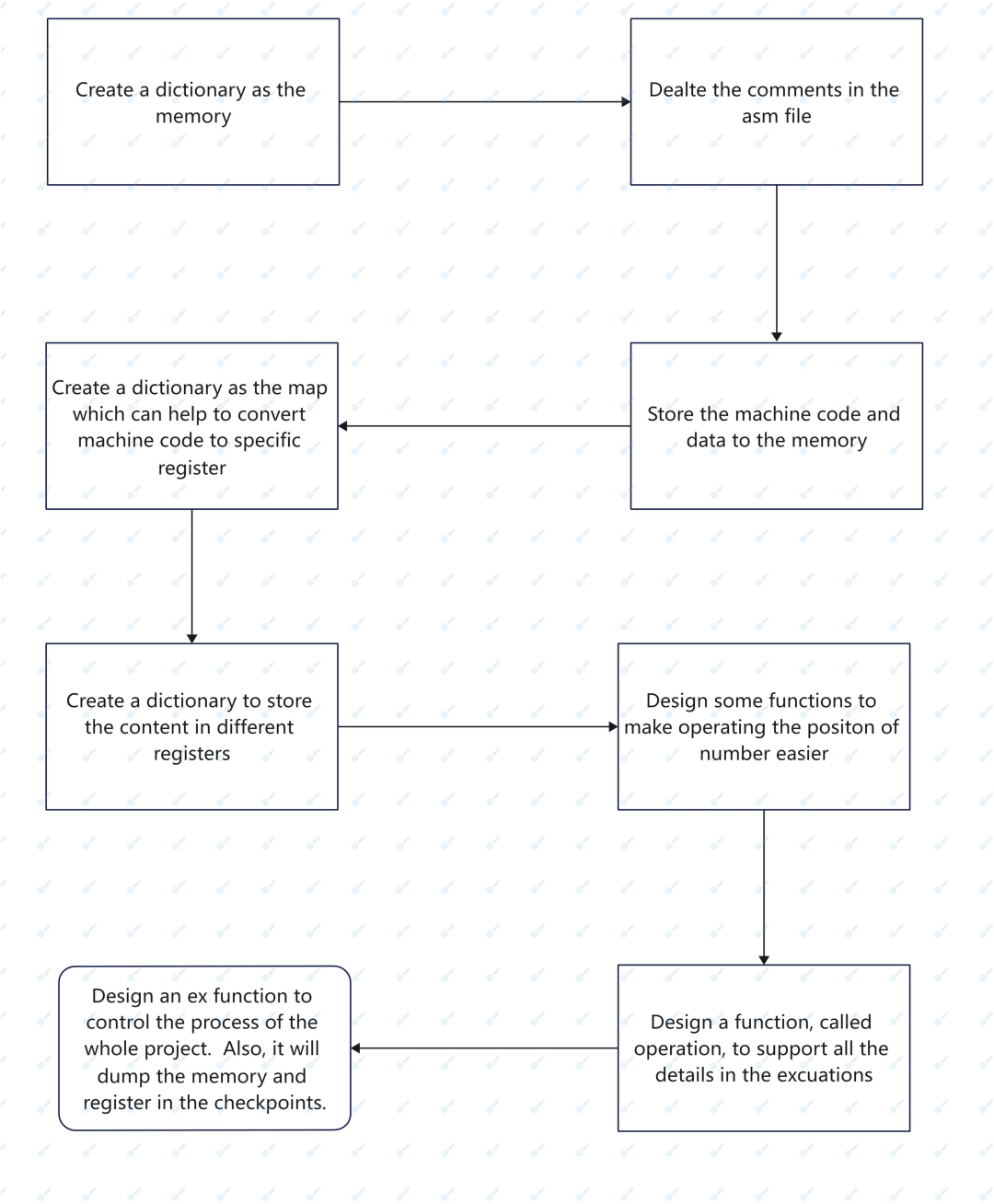
There are another important part in this project--registers. They do not belong to CPU. They are the main reasons that assemblers can support these caculations. They can load ,store, caculate and so on. In order to stimulate the excution, I design some

functions to approach it.

The High Level Implementation Ideas

In order to simulate this process, I divide the whole project into many parts, and then approach them oen by one. Firstly, I design te.py to write some functions to easily to have different positions of a number. Secondly, I design the simulator.py with the help of te.py.

In order to make this whole part more clearly, there is a big picture here.



The graphic of the whole project

Following are the steps of the project:

Step1: Create a dictionary, mem_dic, as the memory, and regard a word as a key.

Step 2: Dealte the comments in the asm file,and store the machine code and data to the mem_dic.

Step 3: Create a dictionary as the map which can help to convert machine code to specific register. Also, create a dictionary to store the content in different registers.

Step 4:Design some functions, such as high8, high16, low8 and so on to make operating the positon of number easier.

Step 5: Design a function, called operation, to support all the details in the excuations.

Step 6: Design an ex function to control the process of the whole project. This function can excute an instruction and then change the pc to the next instruction. Also, it will dump the memory and register in the checkpoints.

Functions in each py_files:

In order to approach these aims, I design many functions in different python file. te:

1. high_8:

Usage: access 8 bits of a number in the high position

Details: the function will use position operation in python to access the 8 high bits in a number.

2. high_mid8:

Usage: access 8 bits of a number in the high_mid position

Details: the function will use position operation in python to access the 8 high_mid bits in a number.

3. low_mid8:

Usage: access 8 bits of a number in the low_mid position

Details: the function will use position operation in python to access the 8 low_mid bits in a number.

4. low_mid8:

Usage: access 8 bits of a number in the low_mid position

Details: the function will use position operation in python to access the 8 low_mid bits in a number.

5. high16:

Usage: access 16 bits of a number in the high position

Details: the function will use position operation in python to access the 16 high bits in a number.

6. low16:

Usage: access 16 bits of a number in the low position

Details: the function will use position operation in python to access the 16 low bits in a number.

simulator:

1. Addbinary:

Usage:to add two number of binary form

Details: this function can add two str binary form to get the result. It return a string.

2. sign_bin:

Usage:get the complement's form of a binary number

Details:this function can easily get the 2's complement form of a binary number. It return a string.

3. create_mem:

Usage: create a dictionary to stimulate the memory.

Details: this function will return a dictionary and regard a word as a key.

4. creat_reg_dic:

Usage:create a dictionary to be a map, which help machine code conver to a name of a register.

5. read_mac_code:

Usage:store the machine code to the the memory.

Details: this function will store the machine code to the dectionary one in a word.

6. read_asm_data:

Usage: store the data to the memory

Details：this function will store the data the number or it’s asci code to the dictionary accroding to the type of the data.

7. reg_con:

Usage: create a dictionart for registers to store some value.

8. no_comments:

Usage: deleate all comments in the asm

Details: this function will delate all the comments in the old asm file, and then store the new content in the oncomment.txt.

9. operation:

Usage: to stimulate different work of the instructions

Details: this function will have different operations accroding to the op or fun in the machine code.

10. write_binary_mc/write_binary:

Usage: dump the memory or register to a file as binary form.

11. ex:

Usage:to excute the whole project and dump the file when it is needed.