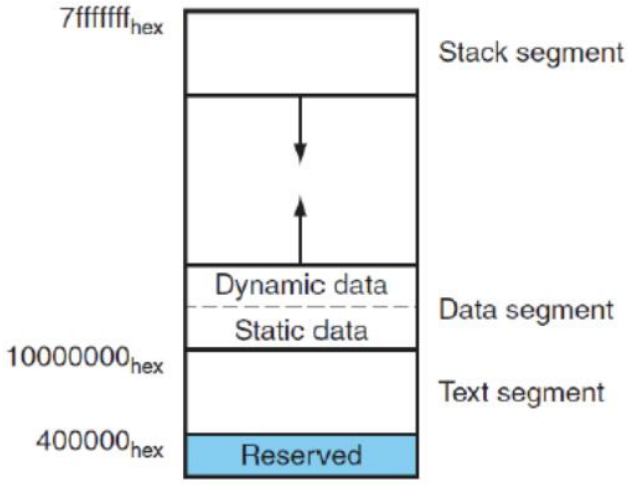
CSC3050 Project1 Report

Liang Mingrui 120090723

2022.3.4

Big Picture Thoughts

MIPS assemblers change MIPS instructions into machine code ,which is consisted of binary number. Firstly, in the whole file, MIPS assembler will only focus on the content under the ".text". Those are the true content that will be change into machine code. And for this homework, the address of these contents will be storged form 0x400000. Like the following picture.

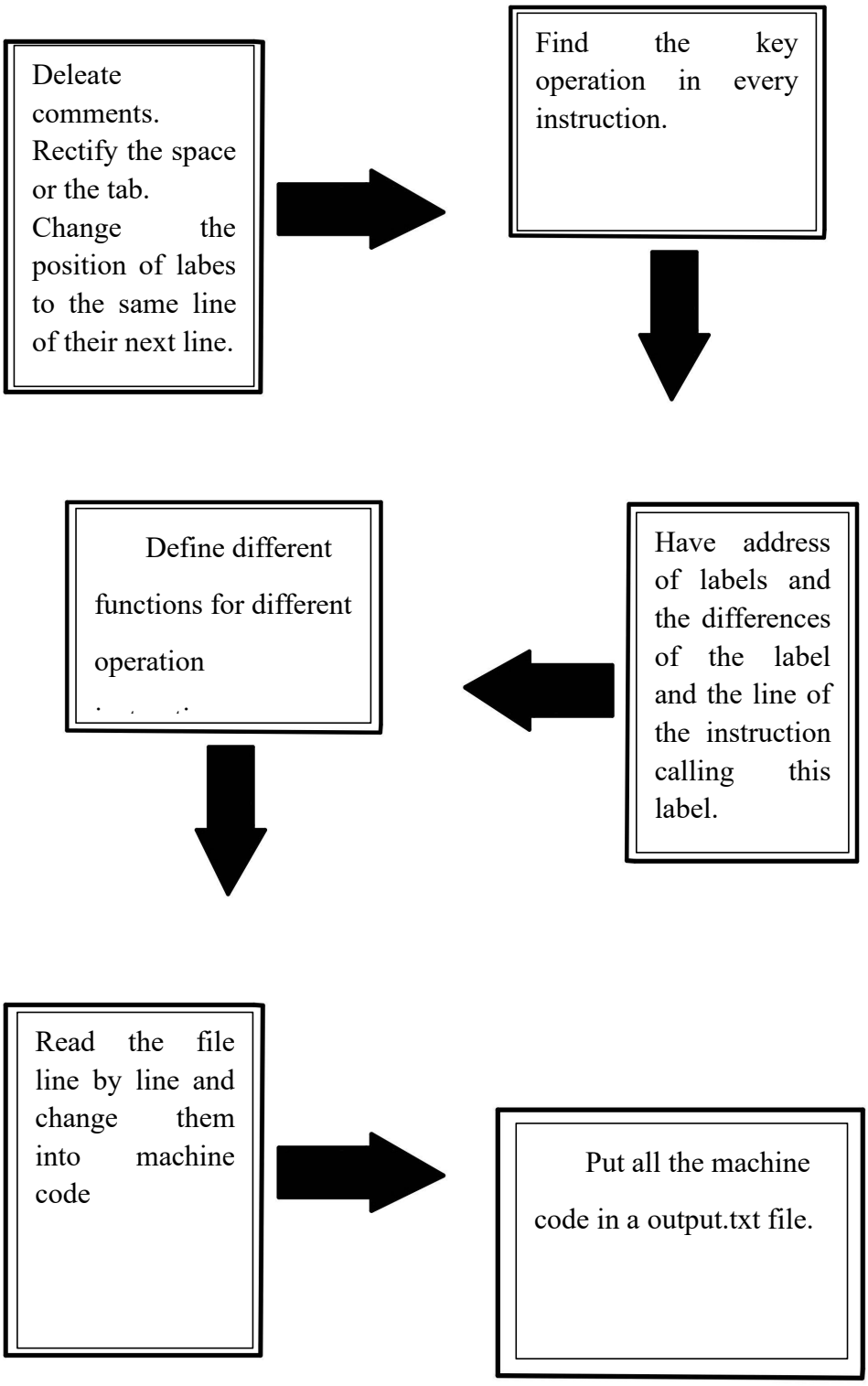


And there are all three kinds of instructions, R, I and J. For a single instruction, it usually is formed by a name of an operation function, a few registers or immediate number. Also for some instructions, label's address is very important. Accroding to their features, I design different functions to read it line by line and change them precisely.

The High Level Implementation Ideas

This is really a big project, so I divide it into many small pieces. There are totaly three py to approach my aim. There are phase1, phase2, and LableTable.

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



Step 1: Deleate all comments, rectify the space or the tab and change the position of labes to the same line of their next line.

Step 2: Find the key operation in every instruction.

Step 3: Have all address of labels and the differences of the line of the label and the line of the instruction calling this label.

Step 4: Define different functions for different operation instructions.

Step 5: Read the file line by line and change them into machine code.

Step 6: Put all the machine code in a output.txt file.

Functions in each py_files:

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

Phase1:

1. no_comment:

Usage: deleate all comments

Details: This function will be offered a name of testfile. It open the test file and then read all its content in a list. Then, it check element one by one whether there is a comment in the line. If there is a comment, it will deleate comment and then store it back to the list. Finally, it will open a new file to store these new line. After that, content without comments will be stored in the new file.

2. fl:

Usage: find the labels in the file

Deails: This function will be offered a string. It select the label judging the symbol ":". Finally, it return the label's name.

LabelTable:

1. stld:

Usage: store all the labels that were found before into a list

Deatails: This function will be offered two list. Then it prepares a empty dictionary first. It will go through two list and matches then. Finally, they will become a dictionary to store label and its coressponding address.

2. address:
Usage:match the label with the corresponding absolute address, and then store them
Details: This function is offered a name of file. It will go through every line in the file. After that, it clean the space or tab first. And then store every line in a absolute address by useing a dictionart. Finally, it return the dictionary.

3. jtl:
Usage: get a 26 bit j type label address
Details: For jump operation, the mips need the 26 bit address. This function first will change absolute address of a label into 32 bit binary number. And then, it delate the PC[31:28] and PC[1:0] to approch the aim. Finally return it.

4. branch_table:
Usage: make a dictionary to store labels and their address for branch
Details: For branch operation, the mips need 16 bit adress. Beacuse its way to get adress is different from the absolute adress, the function need to reopen the file to get new type of address. Finally the function return j as the index of the label, p as the name of the label, and the valid_valid_line as the content.

phase 2
1. addbinary:
Usage: to add binary number
Details: Because the absolute address need to chage into binary form. Also, in case to avoid the negative number, the binary need 2's complement form. The function can add one to the orgin binary number.

2. sign_bin:
Usage: have opposite number to the origin binary number
Details: the function can change every bit in a binary number from 1 to 0 or from 0 to 1. This succed by go through its content one by one.

3. sp_ad:
Usage:specially operation for instructions of loading information from register
Details: For J type instruction, there exist some instructions that need register load something from storage or save something to storage. This kind of instruction is different from others. The functiuon seperate them into valid form.

4. r:
Usage:have final machine code for r type instruction
Details: The function works for R type instruction, it can do different operations accroding to different operations of R instruction. Finally, it return the machine code of the instruction.

5. Ii:
Usage: have final machine code for i type instruction
Details: The function works for I type instruction, it can do different operations accroding to different operations of I instruction. Finally, it return the machine code of the instruction.

6. j:
Usage: have final machine code for i type instruction
Details: The function works for J type instruction, it can do different operations accroding to different operations of J instruction. Finally, it return the machine code of the instruction.

7. ex:
Usage: to use all function to get all machine code and store them in a txt file
Details: This function is a kind of main function, it excuate many functions definding before. It go through all the line and change them into machine code. Finally it write them into a output file and delate all uselss information.

The Implementation Details

1. The key word of operation is stored as lists

```
R = ['add','addu','and','div','divu','jalr','jr','mfhi','mflo','mthi','mtlo','mult',
     'multu','nor','or','sll','sllv','slt','sltu','sra','srav','srl','srlv','sub',
      'subu','syscall','xor']
I = ['addi','addiu','andi','beq','bgez','bgtz','blez','bltz','bne','lb','lbu','lh',
     'lhu','lui','lw','ori','sb','slti','sltiu','sh','sw','xori','lwl','lwr','swl',
     'swr']
J = ['j','jal']
```

2. Register and it's binary number is stored in dictionary

```
reg = {'$zero':'00000','$at':'00001','$v0':'00010','$v1':'00011','$a0':'00100',
       '$a1':'00101','$a2':'00110','$a3':'00111','$t0':'01000','$t1':'01001',
       '$t2':'01010','$t3':'01011','$t4':'01100','$t5':'01101','$t6':'01110',
       '$t7':'01111','$s0':'10000','$s1':'10001','$s2':'10010','$s3':'10011',
       '$s4':'10100','$s5':'10101','$s6':'10110','$s7':'10111','$t8':'11000',
       '$t9':'11001','$k0':'11010','$k1':'11011','$gp':'11100','$sp':'11101',
       '$fp':'11110','$ra':'11111'}
```

3. Labels' address of jume and branch are stored in dictionary(But I did not print them)
4. The final result is stored in the output.txt file

Tips:
I change the tester.py file a little bit.

When excuating the tester.py file, after input the name of the test file, the program will excuate the ex function, and then create a output.txt file. The next input should be the file.