Project1 Report

118010317

Instruction

My project is composed of three source files and one testing file.

phase1.py

labelTabel.py

phase2.py

test.py

Phase 1.py & labelTable.py

This part mainly to read the input file, delete useless comment or symbols, leaving only necessary code for machine to read. The output of this part will be two dictionary: Instr_dict{} and label_dict{}. The keys in label_dict are labels shown in input file, the values are the corresponding address of the label. The keys in instr_dict are address of instructions, and the values are strings of instructions. Remark: the first address is binary of 0x400000 as mentioned in the assignment requirement.

```
Eg. instr_dict = {address , ‘add $t0, $t1, $t’}
Eg. Label_dict = {Fibonacci, address}
```

Implementation

1. delete_comment()   : delete the comments “’ “’ and ###
2. get_label()   : return the label, else will return “
3. get_instr()   : return the instruction
4. hex2bin()   : hexadecimal to binary
5. set_instr()   : read input file, return Instr_dict{} and label_dict{}

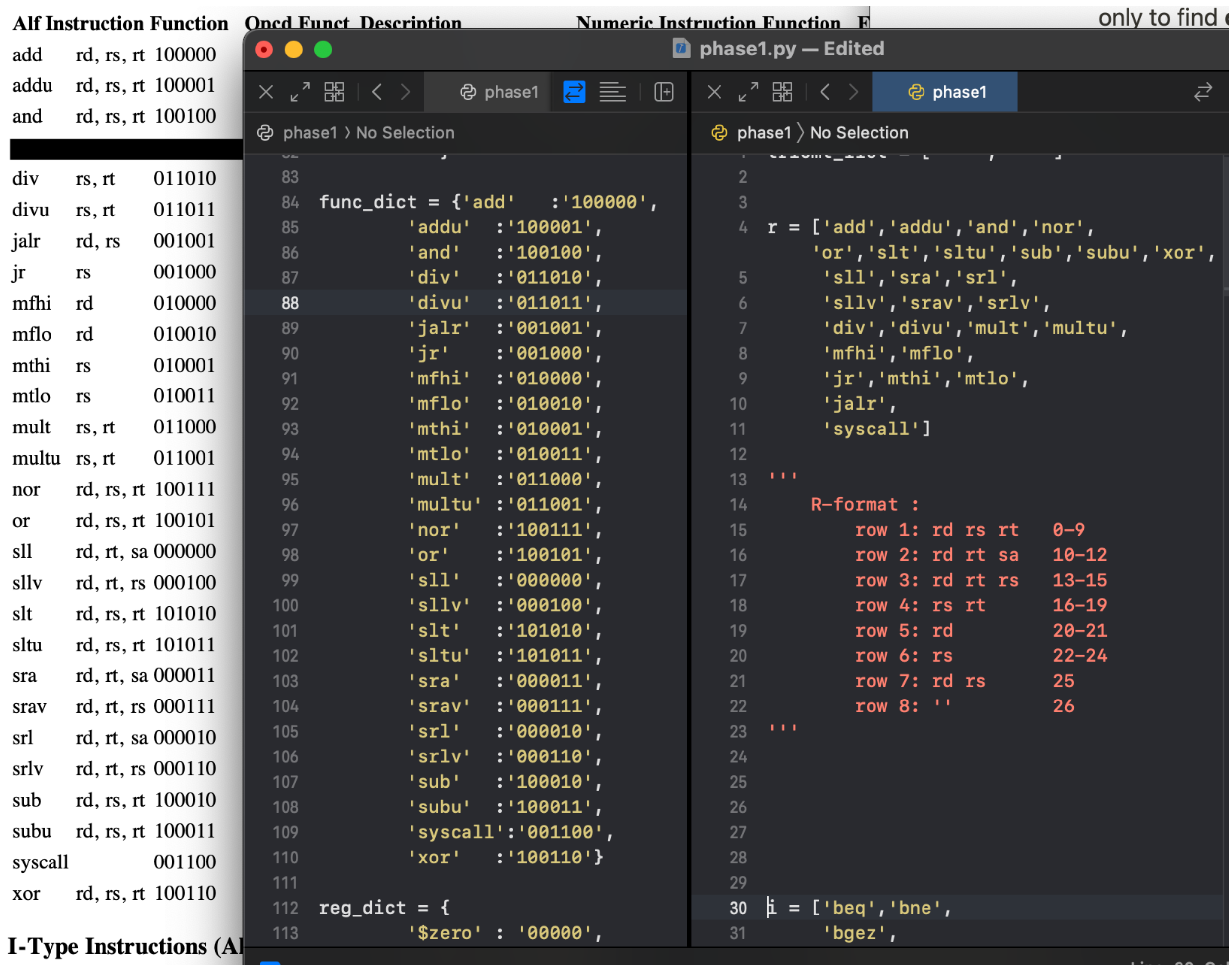
Phase2.py:

R-Type instructions:

According the book, the structure of machine code for an R-format instruction is fixed, the output machine code is expected to have the form:

opcode (6)	rs (5)	rt (5)	rd (5)	sa (5)	function (6)

Therefore, in order to output the machine code for an R-format instruction, we need only to find **opcode, rs, rt, rd, sa, function,** respectively. All R-type instructions use a 000000 **opcode**. The operation is specified by the function field. code for **function** is stored in a dictionary, as shown below.The tricky part is to find **rs, rt, rd, sa.** To do so, I have classify the R-type format instruction into 8 categories. As shown below, for example, the original instruction form in first line in r_list should be `opc rd rs rt`. After defining the type in r-format, I will be able to get rs, rd, sa from the original instruction.



Alf Instruction Function Opcd Funct Description Numeric Instruction Function F

only to find

add	rd, rs, rt	100000
addu	rd, rs, rt	100001
and	rd, rs, rt	100100
div	rs, rt	011010
divu	rs, rt	011011
jalr	rd, rs	001001
jr	rs	001000
mfhi	rd	010000
mflo	rd	010010
mthi	rs	010001
mtlo	rs	010011
mult	rs, rt	011000
multu	rs, rt	011001
nor	rd, rs, rt	100111
or	rd, rs, rt	100101
sll	rd, rt, sa	000000
sllv	rd, rt, rs	000100
slt	rd, rs, rt	101010
sltu	rd, rs, rt	101011
sra	rd, rt, sa	000011
srav	rd, rt, rs	000111
srl	rd, rt, sa	000010
srlv	rd, rt, rs	000110
sub	rd, rs, rt	100010
subu	rd, rs, rt	100011
syscall		001100
xor	rd, rs, rt	100110

I-Type Instructions (Al

```python
func_dict = {'add'    :'100000',
        'addu'  :'100001',
        'and'   :'100100',
        'div'   :'011010',
        'divu'  :'011011',
        'jalr'  :'001001',
        'jr'    :'001000',
        'mfhi'  :'010000',
        'mflo'  :'010010',
        'mthi'  :'010001',
        'mtlo'  :'010011',
        'mult'  :'011000',
        'multu' :'011001',
        'nor'   :'100111',
        'or'    :'100101',
        'sll'   :'000000',
        'sllv'  :'000100',
        'slt'   :'101010',
        'sltu'  :'101011',
        'sra'   :'000011',
        'srav'  :'000111',
        'srl'   :'000010',
        'srlv'  :'000110',
        'sub'   :'100010',
        'subu'  :'100011',
        'syscall':'001100',
        'xor'   :'100110'}

reg_dict = {
        '$zero' : '00000',
```

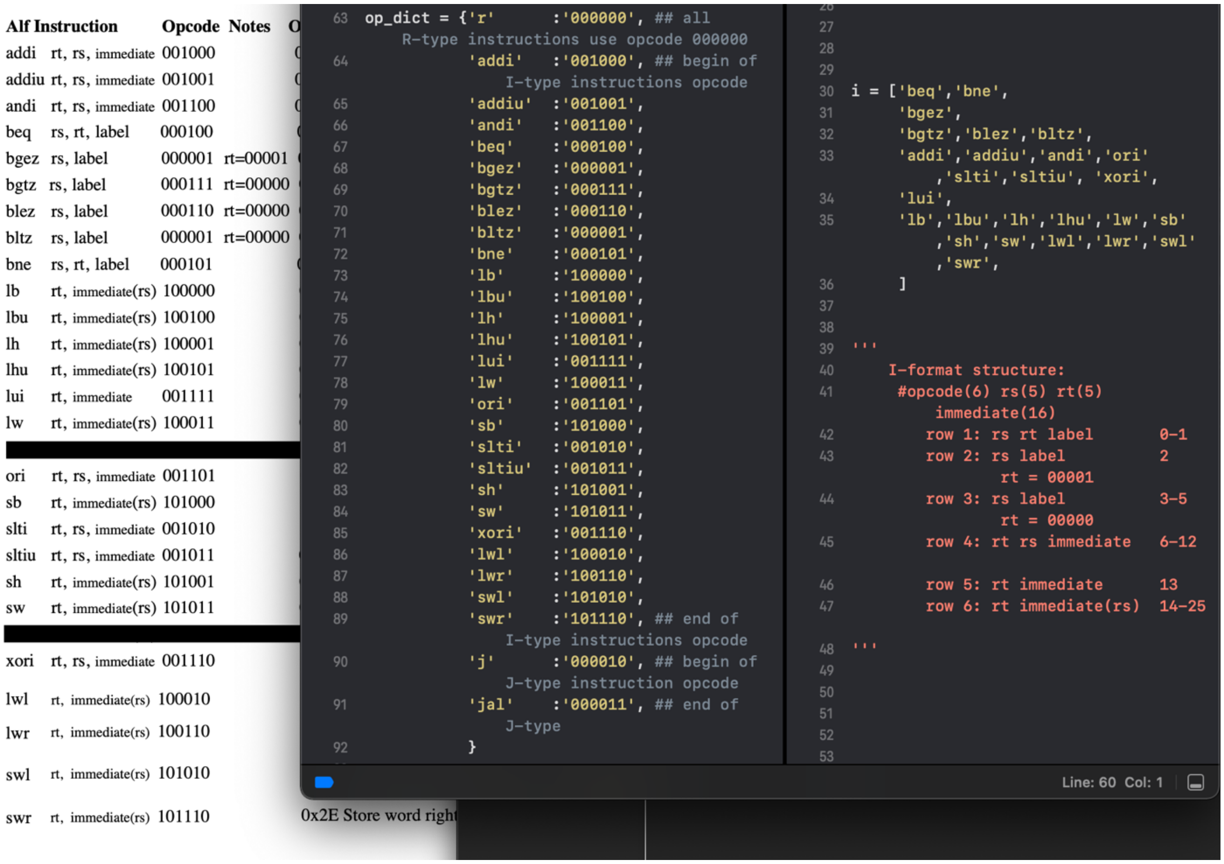
```python
r = ['add','addu','and','nor',
    'or','slt','sltu','sub','subu','xor',
     'sll','sra','srl',
     'sllv','srav','srlv',
     'div','divu','mult','multu',
     'mfhi','mflo',
     'jr','mthi','mtlo',
     'jalr',
     'syscall']

'''
    R-format :
        row 1: rd rs rt    0-9
        row 2: rd rt sa    10-12
        row 3: rd rt rs    13-15
        row 4: rs rt       16-19
        row 5: rd          20-21
        row 6: rs          22-24
        row 7: rd rs       25
        row 8: ''          26
'''







i = ['beq','bne',
     'bgez',
```

I-type Instruction:

According the book, the structure of machine code for an I-format instruction is fixed, the output machine code is expected to have the form:

opcode (6)	rs (5)	rt (5)	immediate (16)

Therefore, in order to output the machine code for an I-format instruction, we need only to find **opcode, rs, rt, rd, immediate** respectively. Code for **function** is stored in a dictionary, as shown below. The tricky part is to find rs, rt, immediate**.** To do so, I have classify the I-type format instruction into 6 categories. As shown below, for example, the original instruction form in first line in r_list should be `opc rs rt label`. After defining the type in i-format, I will be able to get **rs, rt, immediate, label** from the original instruction. However, note that label should be translated to immediate value(16 bit).



Alf Instruction		Opcode	Notes
addi	rt, rs, immediate	001000	
addiu	rt, rs, immediate	001001	
andi	rt, rs, immediate	001100	
beq	rs, rt, label	000100	
bgez	rs, label	000001	rt=00001
bgtz	rs, label	000111	rt=00000
blez	rs, label	000110	rt=00000
bltz	rs, label	000001	rt=00000
bne	rs, rt, label	000101	
lb	rt, immediate(rs)	100000	
lbu	rt, immediate(rs)	100100	
lh	rt, immediate(rs)	100001	
lhu	rt, immediate(rs)	100101	
lui	rt, immediate	001111	
lw	rt, immediate(rs)	100011	
ori	rt, rs, immediate	001101	
sb	rt, immediate(rs)	101000	
slti	rt, rs, immediate	001010	
sltiu	rt, rs, immediate	001011	
sh	rt, immediate(rs)	101001	
sw	rt, immediate(rs)	101011	
xori	rt, rs, immediate	001110	
lwl	rt, immediate(rs)	100010	
lwr	rt, immediate(rs)	100110	
swl	rt, immediate(rs)	101010	
swr	rt, immediate(rs)	101110	

0x2E Store word right

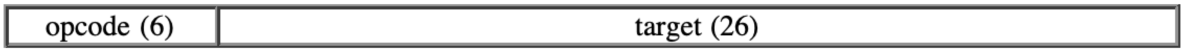
```
op_dict = {'r'      :'000000', ## all R-type instructions use opcode 000000
           'addi'   :'001000', ## begin of I-type instructions opcode
           'addiu'  :'001001',
           'andi'   :'001100',
           'beq'    :'000100',
           'bgez'   :'000001',
           'bgtz'   :'000111',
           'blez'   :'000110',
           'bltz'   :'000001',
           'bne'    :'000101',
           'lb'     :'100000',
           'lbu'    :'100100',
           'lh'     :'100001',
           'lhu'    :'100101',
           'lui'    :'001111',
           'lw'     :'100011',
           'ori'    :'001101',
           'sb'     :'101000',
           'slti'   :'001010',
           'sltiu'  :'001011',
           'sh'     :'101001',
           'sw'     :'101011',
           'xori'   :'001110',
           'lwl'    :'100010',
           'lwr'    :'100110',
           'swl'    :'101010',
           'swr'    :'101110', ## end of I-type instructions opcode
           'j'      :'000010', ## begin of J-type instruction opcode
           'jal'    :'000011', ## end of J-type
           }
```

```
i = ['beq','bne',
     'bgez',
     'bgtz','blez','bltz',
     'addi','addiu','andi','ori'
         ,'slti','sltiu', 'xori',
     'lui',
     'lb','lbu','lh','lhu','lw','sb'
         ,'sh','sw','lwl','lwr','swl'
         ,'swr',
     ]


'''
    I-format structure:
     #opcode(6) rs(5) rt(5)
         immediate(16)
        row 1: rs rt label       0-1
        row 2: rs label          2
                rt = 00001
        row 3: rs label          3-5
                rt = 00000
        row 4: rt rs immediate   6-12

        row 5: rt immediate      13
        row 6: rt immediate(rs)  14-25

'''
```

Line: 60 Col: 1

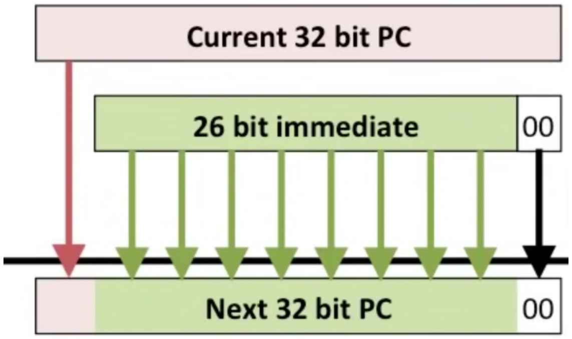
J-type Instruction:

According the book, the structure of machine code for an J-format instruction is fixed, the output machine code is expected to have the form:



opcode (6)	target (26)

Therefore, in order to output the machine code for an I-format instruction, we need only to find **opcode, target** respectively. Code for **function** is stored in a dictionary, as shown below. The tricky part is to find target (26 bit)**.**



Implementation:

1. classify_type() : return the instruction type: R or I or J
2. get_rstrc() : return the classification result in R-type
3. get_istrc() : return the classification result in I-type
4. get_op() : return the opcode of R-type, I-type, J-type respectively
5. get_rs() : return register rs in R-type, I-type repectively
6. get_rt() : return register rt in R-type, I-type repectively
7. get_rd() : return register rd in R-type
8. get_shamt() : return shift amount of R-type
9. get_func() : return function in R-type
10. get_imdt() : return immediate value(16 bit) in I-type
11. get_target() : return target value(26 bit) in J-type
12. num2bin() : return positive integer number to binary case
13. neg2bin() : return negative integer number to binary case

Testing:

I use both the .asm file and get the output .txt file. After comparing with the expected .txt file, I have passed the test and get all the answers right.