Lab6 Report

1. Algorithm explanation

First, we need to use input to get the data from stdin. We use a iteration and stop when we get '.end'. We will store the input all in list **read.** Second, we need to traverse **read** for the first time to construct the label table, which be stroed in dictionary **label**. The key is the identifier of the label and the value is the address of the label. Attention that not every instruction will occupy only one address. Instructions like .stringz and .blkw will occupy more addresses. Third we need to traverse **read** secondly, this time we will print the binary instruction after we assembling one line.

2. Essential codes (every part counts)

```
reg =
{'r0,':'000','r1,':'001','r2,':'010','r3,':'011','r4,':'100','r5,':'101
','r6,':'110','r7,':'111','r0':'000','r1':'001','r2':'010','r3':'011','
r4':'100','r5':'101','r6':'110','r7':'111'  }#transfer reg to binary
dic =
{'0':0,'1':1,'2':2,'3':3,'4':4,'5':5,'6':6,'7':7,'8':8,'9':9,'a':10,'b'
:11,'c':12,'d':13,'e':14,'f':15} #transfer hex to dec
instructions =
['add','and','not','ld','ldr','ldi','lea','st','str','sti','trap','br',
'jmp','jsr','rti','.orig','.fill','.blkw','.stringz','.end','getc','out
','puts','in','putsp','halt','brn','brz','brp','brnz','brnp','brzp','br
nzp','ret','jsrr']#all the instructions
#we will call the function twice when we meet the .stringz
#first time we will meet it when we are making the tabel
#which means we dont need to print anything at this time so we will
call with prt = 0
#but when we meet stringz in the second traverse, we need print,so we
call with prt = 1
def stringpro(ind,prt=0,flag=0):
    for char in line:
        if char == '"':
            flag += 1
        if flag == 1 :
            ind += 1
            if ( prt == 1 ) & (char != '"'):
                print(dectobin(ord(char)))
    if prt == 1:
        print("0000000000000000")
    return ind
#this function will transfer hex like 'x20' 'x123' to 'x0020' and
'x0123'
#we need an argument of the string of the former hex and return the
standard hex
def extendhex(hex,subj='x'):
```

```
    if hex[1] == '-':
        lent = len(hex[2:])
        last = hex[2:]
        subj += '-'
    else:
        lent = len(hex[1:])
        last = hex[1:]
    for number in range(4-lent):
        subj +='0'
    return subj + last
#this function will test the br series instructions and return the
binary instrction string
#we need to add 'nzp' by checking if they are in the instruction
#finally we will combine three parts of the binary instruction
def test(part,nzp =''):
    for char in ['n','z','p']:
        if char in part[0]:
            nzp += '1'
        else:
            nzp += '0'
    if nzp == '000':
        nzp = '111'
    return '0000' + nzp + off(9,part[1])
#this function will get the argument of the last part of add or
and ,then it will return the binary part of it
#if the first char in this part is 'r' then its a reg mode
#if not ,then it is the imm mode
def adder(imm):
    if imm[0] == 'r':
        return '000' + reg[imm]
    return '1' + imme(5,imm)
#this function will get the number of digits and the "#1234" or "x1234"
string , then return the binary string of the imm
#first we get the number part by imm[1:] then use functions to transfer
it to binary then get the last part
def imme(digit,imm):
    if imm[0] == '#':
        return dectobin(int(imm[1:]))[-digit:]
    return dectobin(hextodec(extendhex(imm)))[-digit:]
#this functions will get the number of digits and the '#1234' or
'x1234' string, then return the binary string of the off
def off(digit,off):
    if off[0] == '#':
        return dectobin(int(off[1:]))[-digit:]
```

```
    return dectobin(label[off] - inde)[-digit:]
#this function will transfer hex to decimal ,in account for the sign
def hextodec(hex):
    if hex[1] == '-':
        return -(dic[hex[2].lower()]*(16**3) +
dic[hex[3].lower()]*(16**2) + dic[hex[4].lower()]*(16) +
dic[hex[5].lower()]*(1))
    return dic[hex[1].lower()]*(16**3) + dic[hex[2].lower()]*(16**2) +
dic[hex[3].lower()]*(16) + dic[hex[4].lower()]*(1)
#this function will transfer decimal to binary ,in account for the
sign. We get the negtive number by adding it with 65536
def dectobin(dec,bin=''):
    if dec < 0:
        dec += 65536
    #list 32768 16384 8192 4096 2048 ..... 4 2 1
    for number in [ int(1/2 ** value) for value in range(-15,1)]:
        bin += str(dec//(number))
        dec %= number
    return bin
#initialize all the parameters
read,line,label,inde,index = [],'',{},0,0
#use the iteration to get the input ,store it in read
while line.strip().lower() != '.end':
    line = input()
    read.append(line.strip())
#the first traverse,we first split it ,transfer to lower case ,and we
will skip the empty line
for line in read:
    part = line.split()
    if not part:
        continue
    for number in range(len(part)):
        part[number] = part[number].lower()
#if it is .orig ,we will set the index for the second traverse and the
inde for the first traverse
    if part[0] == '.orig':
        index = hextodec(part[1])
        inde = hextodec(part[1]) - 1
        continue
#if its first part is not an instruction ,it is a label. Store it and
remove the first part , which is the label
    if part[0] not in instructions:
        label[part[0]] = index
        part = part[1:]
```

```
#to see how many memory it will occupy
    if part[0] == '.blkw' :
        index += int(part[1][1:])
    elif part[0] == '.stringz':
        index = stringpro(index)
    else:
        index += 1
 #the second traverse, the former part is the same
for line in read:
    part = line.split()
    if not part:
        continue
    for number in range(len(part)):
        part[number] = part[number].lower()
    if part[0] not in instructions:
        part = part[1:]
    inde += 1
#process every instruction respectively
    if part[0] == '.blkw':
        for number in range(int(part[1][1:])):
            print('0111011101110111')
        inde += int(part[1][1:]) - 1
    if part[0] in ['br' , 'brn' , 'brz' , 'brp' , 'brnp' , 'brnz' ,
'brzp' , 'brnzp']:
        print(test(part))
    if part[0] == '.stringz':
        inde = stringpro(inde,prt=1)
    if part[0] == '.fill':
        print(imme(16,part[1]))
    if part[0] == 'add':
        print('0001' + reg[part[1]] + reg[part[2]] + adder(part[3]))
    if part[0] == 'and':
        print('0101' + reg[part[1]] + reg[part[2]] + adder(part[3]))
    if part[0] == 'not':
        print('1001' + reg[part[1]] + reg[part[2]] + '111111')
    if part[0] == 'ld':
        print('0010' + reg[part[1]] +off(9,part[2]))
    if part[0] == 'ldr':
        print('0110' + reg[part[1]] + reg[part[2]] +imme(6,part[3]))
    if part[0] == 'ldi':
        print('1010' + reg[part[1]] + off(9,part[2]))
    if part[0] == 'lea':
        print('1110' + reg[part[1]] + off(9,part[2]))
    if part[0] == 'st':
```

```
        print('0011' + reg[part[1]] + off(9,part[2]))
    if part[0] == 'str':
        print('0111' + reg[part[1]] + reg[part[2]] +imme(6,part[3]))
    if part[0] == 'jsr':
        print('01001' +off(11,part[1]))
    if part[0] == 'sti':
        print('1011' + reg[part[1]] +off(9,part[2]))
    if part[0] == 'trap':
        print('11110000' + dectobin(hextodec(extendhex(part[1])))[-8:])
    if part[0] == 'jmp':
        print('1100000' + reg[part[1]] + '000000')
    if part[0] == 'jsrr':
        print('0100000' + reg[part[1]] + '000000')
    if part[0] == 'ret':
        print('1100000111000000')
    if part[0] == 'rti':
        print('1000000000000000')
    if part[0] == '.orig':
        print(dectobin(hextodec(part[1])))
    if part[0] == 'getc':
        print('1111000000100000')
    if part[0] == 'out':
        print('1111000000100001')
    if part[0] == 'puts':
        print('1111000000100010')
    if part[0] == 'in':
        print('1111000000100011')
    if part[0] == 'putsp':
        print('1111000000100100')
    if part[0] == 'halt':
        print('1111000000100101')
```