w6: Computer Architecture - Project2

Date	@October 19, 2022
Status	Not started
Type	Assignment
倒數時間	還剩 -4 天
Due date	@November 16, 2022

程式說明：

```
respond = []
sub = []
#instructions
operation = ['add','sub','lw','sw','addi','addu','addiu','sll','srl','and','or','nor',
             'beq','bne','slt','slti','sltu','j','jal','jr']
#zreo,for IType const,arguments,temp,contents for later,temp,OS,...
reg = {'$zero': 0,
        '$at':None,
        '$v0':None,'$v1':None,
        '$a0':None,'$a1':None,'$a2':None,'$a3':None,
        '$t0':None,'$t1':None,'$t2':None,'$t3':None,'$t4':None,'$t5':None,'$t6':None,'$t7':None,
        '$s0':None,'$s1':None,'$s2':None,'$s3':None,'$s4':None,'$s5':None,'$s6':None,'$s7':None,
        '$t8':None,'$t9':None,
        '$k0':None,'$k1':None,
        '$gp':None,'$sp':None,'$fp':None,'$ra':None }
reg_table = {'$0':'$zero',
            '$1':'$at',
            '$2':'$v0','$3':'$v1',
            '$4':'$a0','$5':'$a1','$6':'$a2','$7':'$a3',
            '$8':'$t0','$9':'$t1','$10':'$t2','$11':'$t3','$12':'$t4','$13':'$t5','$14':'$t6','$15':'$t7',
            '$16':'$s0','$17':'$s1','$18':'$s2','$19':'$s3','$20':'$s4','$21':'$s5','$22':'$s6','$23':'$s7',
            '$24':'$t8','$25':'$t9',
            '$26':'$k0','$27':'$k1',
            '$28':'$gp','$29':'$sp','$30':'$fp','$31':'$ra'}
```

全域變數有operation、reg，和reg_table等，reg_table是暫存器的對照表，reg負責存暫存器的值。

```
def parse(User_input):           #split and classify line to op or reg
    tokens=[]
    splitedLine = re.split('[#:(,\s]+',User_input)
    for c in splitedLine:
        if c != "":
            if c.lower() in operation:
                tokens.append([c,"OPRT"])
            elif isNum(c):
                c = int(c)
                tokens.append([c,"NUM"])
            elif c in reg_table or c in reg_table.values():
                c = c.strip( '$' )
                tokens.append([c,"REG"])
            else:
                tokens.append([c,"Unknown"])
        else:
            return False
    return tokens
```

parse()是負責分字和分類詞的類別，方便接下來operation確認正確的格式並且適當回報錯誤訊息。

```
def executeInstr(tokens):
    global respond
    respond = []
    if tokens==False:return False

    if(tokens[0][1]=="OPRT"):
        if len(tokens) >= 4:
            if(tokens[0][0].lower() == "add"):
                add(tokens)
            elif(tokens[0][0].lower() == "sub"):
                sub(tokens)
            elif(tokens[0][0].lower() == "lw"):
                lw(tokens)
            elif(tokens[0][0].lower() == "sw"):
                sw(tokens)
            elif(tokens[0][0].lower() == "addi"):
```

executeInstr()根據parse()切好的詞組數量和第一個詞(通常為operation)呼叫對應實作的function，若第一個詞並非operation則回報錯誤，或

```
                addi(tokens)
            elif(tokens[0][0].lower() == "addu"):
                addu(tokens)
            elif(tokens[0][0].lower() == "addiu"):
                addiu(tokens)
            elif(tokens[0][0].lower() == "sll" or tokens[0][0].lower() == "srl"):
                shift(tokens)
            elif(tokens[0][0].lower() == "and" or tokens[0][0].lower() == "or" or tokens[0][0].lower() == "nor"):
                AndOrNor(tokens)
            elif(tokens[0][0].lower() == "beq" or tokens[0][0].lower() == "bne"):
                branch(tokens)
            elif(tokens[0][0].lower() == "slt"):
                slt(tokens)
            elif(tokens[0][0].lower() == "slti"):
                slti(tokens)
            elif(tokens[0][0].lower() == "sltu"):
                sltu(tokens)
        elif len(tokens) == 2:
            if(tokens[0][0].lower() == "j" or tokens[0][0].lower() == "jal" or tokens[0][0].lower() == "jr"):
                jump(tokens)
        else:
            respond.append(["ERR",f"Error:Expected identifier."])
    else:
        respond.append(["Error",f"Unexpected Operation：{tokens[0][0]}."])
```

詞組數量不符，例如：缺少暫存器，也回報錯誤。

```
def lw(tokens):
    global respond
    stopExe = False

    for i in range(1,4,1):
        if (i%2==1)and(tokens[i][1] != "REG"):
            respond.append(["Error",f"Malformed {tokens[0][0]} statement, no suitable register {tokens[i][0]}."])
            stopExe = True
        elif(tokens[i][1] == "NUM")and(tokens[i][0] > 65535 or tokens[i][0] < 0):
            respond.append(["Error",f"Malformed {tokens[0][0]} statement, {tokens[i][0]} is out of range."])
            stopExe = True
        elif (tokens[i][1] == "Unknown"):
            respond.append(["Error",f"Unknown identifier {tokens[i][0]} ."])
    respond.append(["operation",tokens[0][0]])
    if stopExe == False:
        respond.append(["source 1",f"${tokens[3][0]}"])
        address = f"MEM[${tokens[3][0]}+{tokens[2][0]}]"
        respond.append(["memory address",address])
        respond.append(["desstination",f"${tokens[1][0]}"])
        Itype("100011",tokens[3][0],tokens[1][0],tokens[2][0],1)
```

當operation為lw時，executeInstr() 會呼叫lw()。

首先會先確認operand的類別是否正確，還有立即值是否有超出範圍，若有就將錯誤回報資訊存入respond。

而不管operand是否有錯誤，operation都會回傳，所以將operation的類別存入respond。

最後，如果instruction都沒有回報錯誤，則會將source、destination，和memory address等資訊存入respond中，並且呼叫Itype()。

以下function皆會確認operand、回報錯誤，和呼叫Type：

add、sub、lw、sw、addi、addu、addiu、shift、AndOrNor、branch、 slt、slti、sltu、jump

```
def Itype(op,rs,rt,imm,flag):
    respond.append(["opcode",op])
    respond.append(["rs",rs])
    respond.append(["rt",rt])
    if flag==0:
        respond.append(["offset",imm])
    else:
        respond.append(["immediate value",imm])
```

根據instruction回傳的opcode轉成輸出的格式。

instruction不同，會呼叫不同type，除了Itype，還有：

Rtype(op,rs,rt,rd,shamt,funct)、Jtype(op,address)

```
# GUI
User_input=''
window = tk.Tk()

width = 960
height = 540
```

```
window_width = window.winfo_screenwidth()
window_height = window.winfo_screenheight()
left = int((window_width - width)/2)
top = int((window_height - height)/2)
Str_Len = 30

window.title('MIPS instruction decoder 2')
window.geometry(f'{width}x{height}+{left}+{top}') # window show size
window.configure(bg='#fff') # window BG color
label = tk.Label(window, bd=10, bg='#ffa',fg='#000', text = "MIPS instruction decoder").pack(side="top",fill="x") # Title Label

# input block
Input_label = tk.Label(window, bg='#fff', text='Please enter MIPS instruction:',font=("Times",14,"bold"))
Input_label.place(relx=0.2,y=100)

entry = tk.Entry(window,width=Str_Len,font=("Times",16))
entry.place(relx=0.6,y=100)

button = tk.Button(window, width=8, height=1,bg="#ffa", command=lambda:show(entry),text='Decode', font=("Times",10)) # button
button.place(relx=0.5,y=140,anchor = 'center')


def show(self):
    global User_input,respond
    User_input = self.get()
    User_input = ''.join(User_input.replace(' ',',',1))
    User_input = User_input.strip( ')' )
    executeInstr(parse(User_input))

    if type(respond) != str:
        text = tk.Text(window,height=10, font=("Times",14,"bold"))
        for res in respond:
            text.insert(tk.INSERT,f"{res[0]}: {res[1]}\n")
        text.place(relx=0.2,rely=0.5)
#         Out_label = tk.Label(window , text=Txt, font=("Times",14,"bold"), bd=10 , fg='#225', bg= "#fff",justify = 'left',anchor = 'w
#         Out_label.place(relx=0.2,rely=0.5)

    else:
         print(type(respond))
#         Out_label.configure(text=f"{ans}",bd=152 , fg='#225', bg= "#aaa",anchor = 'center')
    return respond

window.mainloop()
print(User_input)
```

顯示GUI的部分。

介面包含輸入列、指示文字、按鈕和輸出框，按下按鈕時會執行show()。

show()會將輸入列的文字傳給parse()，並且將respond裡的資料一條條輸出到輸出框上。

心得：

我這次使用的程式語言與上次demo使用不一樣，跟C++相比Python實作起來更容易，我可以花更少的時間debug，花更多時間思考function之間的運作和資料傳遞和GUI上，除此之外這次花更多的時間注意各種不同operation的細節。