```python
from instructions import inst as Instruction


class AssemblyTranslator:

    __machineLang = []                      #all instructions in assembly that translated to machine language
    __fillValue = []                        #collect all .fill variables and make list of pair [symbolic,values]
    __assembly = []                         #save all instructions in assembly line by line
    __labelList = 0
    __inst = Instruction
    errorDetect = False
    errorDetail = ""

#used functions

    def printer(self, des = "textToSimulator.txt", inputList = __machineLang):          #TODO: write machine language to text file
        file = open(des, "w")                                                            #TODO: "r" - Read - Default value. Opens a file for reading, error if the file does not exist
        for i in inputlist:                                                              #TODO: "a" - Append - Opens a file for appending, creates the file if it does not exist
            file.write(str(i)+"\n")                                                      #TODO: "x" - Create - Creates the specified file, returns an error if the file exists
        file.close()                                                                     #TODO: "w" - Write - Opens a file for writing, creates the file if it does not exist

    def __twosCom_decBin(self, dec, bit):
        if dec >= 0 :
            bin1 = bin(dec).split("0b")[1]
            while len(bin1) < bit:
                bin1 = '0'+bin1
            return bin1
        else:
            bin1 = -1*dec
            return bin(bin1 - pow(2,bit) ).split("0b")[1]

    def checkSameLable(self,List=__labelList):                      #7 check The same Lable in List
        count = 0
        for i in range (0,len(List)) :
            firstLabel = List[i]
            count += 1
            for j in range (count,len(List)):
                if(firstLabel == List[j] and self.errorDetect != True):
                    self.errorDetect = True
                    self.errorDetail = "Same Lable"


    def addLabelinList(self,List=__assembly):                       #7 Add labels to list
        newList = []
        for i in range(len(List)):
            if(List[i][0] !=None):
                newList.append(List[i][0])

        return newList

    def __fillFinding(self):                #TODO: this function should be call first and find .fill in assembly file and return line where .fill is
        for i in self.__assembly:           #TODO: and then collect all .fill variables and make list of pair [symbolic,values]
            if(i[1]=='.fill'):
                sheet =[i[0],i[2]]
                self.__fillValue.append(sheet)


    def __binToDec(self,binary):            #TODO: convert intput num in binary to decimal number
        if(type(binary) is str):
            return int(binary,2)
        else:
            return print('Type of binary not string')


    def translator(self,item):              #TODO: make this function that input item looks like ['start', 'add', '1', '2', '1'] and translate to binary

        textTranslated = ""
        isSymbolic = False

        labels, instcode, regA, regB, destReg = item[0], item[1], item[2], item[3], item[4]
        if (instcode in Instruction["name"]) :
            indexOfInst = self.__inst["name"].index(instcode)
            type = self.__inst["type"][indexOfInst]
            optc_bin = self.__inst["optc_bin"][indexOfInst]

            if type == "R" :
                textTranslated += "0000000"
                textTranslated += optc_bin
                textTranslated += self.__regDecoder3bit(regA)
                textTranslated += self.__regDecoder3bit(regB)
                textTranslated += "0000000000000"
                textTranslated += self.__regDecoder3bit(destReg)

            elif type == "I" :

                if (self.__inst["name"][indexOfInst] == "beq"):
                    textTranslated += "0000000"
                    textTranslated += optc_bin
                    textTranslated += self.__regDecoder3bit(regA)
                    textTranslated += self.__regDecoder3bit(regB)

                    sybolicAddress = ""
                    isSymbolic = False

                    indexOfItem = self.__assembly.index(item)
                    for i in range(len(self.__assembly)):
                        if(self.__assembly[i][0] == destReg):
                            sybolicAddress = str((indexOfItem+1-i)*-1)      #this eqation for calculate how many line should add(sub) in offsetField
                            isSymbolic = True
                            break
                        isSymbolic = False

                else:
                    textTranslated += "0000000"
                    textTranslated += optc_bin
                    textTranslated += self.__regDecoder3bit(regA)
                    textTranslated += self.__regDecoder3bit(regB)

                    sybolicAddress = ""
                    for i in range(len(self.__assembly)):
                        if(self.__assembly[i][0] == destReg):
                            sybolicAddress = str(i)
                            isSymbolic = True
                            break


                if (isSymbolic) :
                    textTranslated += self.__twosCom_decBin(int(sybolicAddress),16)

                elif(not destReg.isdigit()):
                    self.errorDetect = True
                    self.errorDetail = "Using undefined labels"

                else :
                    if(int(destReg)>-32768 and int(destReg) <= 32767):
                        if (int(destReg) < 0) :
                            textTranslated += self.__twosCom_decBin(int(destReg))
                        else :
                            textTranslated += '{0:016b}'.format(int(destReg))
                    else:
                        self.errorDetect = True
                        self.errorDetail ="Out of range destReg more 16 bit"


            elif type == "J" :
                textTranslated += "0000000"
                textTranslated += optc_bin
                textTranslated += self.__regDecoder3bit(regA)
                textTranslated += self.__regDecoder3bit(destReg)
                textTranslated += "0000000000000000"                    #7 Bit 15 - 0 should be zero "0"*16


            elif type == "O" :
                textTranslated += "0000000"                             #7 Bit 24 - 22 opcode
                textTranslated += optc_bin
                textTranslated += "00000000000000000000"                #7 Bit 21 - 0 should be zero "0"*22

                # print(textTranslated)                                 #for debugging purposes
                # print(self.__binToDec(textTranslated))                #for debugging purposes
            self.__machineLang.append(self.__binToDec(textTranslated))

        elif (instcode == ".fill"):                                                 #for .fill
            isSymbolicAddress = False
            for i in range(len(self.__assembly)):
                if (regA == self.__assembly[i][0]):
                    textTranslated = bin(int(i))
                    isSymbolicAddress = True
                    break


            if (isSymbolicAddress == False):
                textTranslated = bin(int(regA))

            # print(textTranslated)                                         #for debugging purposes
            # print(self.__binToDec(textTranslated))                        #for debugging purposes
            self.__machineLang.append(self.__binToDec(textTranslated))


        else:
            self.errorDetect = True
            self.errorDetail = "Using opcode other than those specified"


    def __regDecoder3bit(self, number):                     #TODO: decode reg from dec to bin like from '5' to '101'
        number = int(number)
        if( number >= 0 and number < 8 ):
            if(number==0):
                return "000"
            elif(number==1):
                return "00"+bin(number).replace("0b", "")
            elif(number<4):
                return"0"+bin(number).replace("0b", "")
            elif(number>3):
                return bin(number).replace("0b", "")        #7 https://www.geeksforgeeks.org/python-program-to-covert-decimal-to-binary-number/
        # elif number < 0 and number > -7:
        #   return bin(number if number>0 else number+(1<<3)).replace("0b", "")
        else:
            self.errorDetect = True
            self.errorDetail = "out of range 3 bit"
            return""


    def __simplify(self,listTransformed):           #TODO: this function should delete all comments and formating

        resList = []
        for item in listTransformed:
            if (item[0] in Instruction["name"]):    #check instions of this line have the same name as the instruction in the instruction list
                if (item[0]  == "halt"):            #and spacial case for "halt" it can be in both item[0] and item[1]
                    labels, instcode, regA, regB, destReg = None, item[0], None, None, None
                elif item[0] == "noop":
                    labels, instcode, regA, regB, destReg = None, item[0], None, None, None
                elif item[1] == "noop":
                    labels, instcode, regA, regB, destReg = item[0], item[1], None, None, None
                elif item[1] == "halt":
                    labels, instcode, regA, regB, destReg = item[0], item[1], None, None, None
                elif item[0] == "jalr":
                    labels, instcode, regA, regB, destReg = None, item[0], item[1], None, item[2]
                elif item[1] == "jalr":
                    labels, instcode, regA, regB, destReg = item[0], item[1], item[2], item[3], None
                else:
                    labels, instcode, regA, regB, destReg = None, item[0], item[1], item[2], item[3]       #if have the same name as the instruction in the instruction list
            elif (item[1] == ".fill") :
                labels, instcode, regA, regB, destReg = item[0], item[1], item[2], None, None

            else:
                errorDetect = True
                errorDetail = "Using opcode other than those specified"
                labels, instcode, regA, regB, destReg = item[0], item[1], item[2], item[3], item[4]

            sheet = [labels, instcode, regA, regB, destReg]     #contains data in instruction list format
            resList.append(sheet)

        self.__labelList= self.addLabelinList(resList)
        self.checkSameLable(self.__labelList)                               # checkSameLable
        # print(self.__labelList)
        return resList


    # def stringReader(self,filelocation = "assembler\demofile copy.txt"):
    def stringReader(self,filelocation = "assembler\demofile.txt"):

        f = open(filelocation, "r")
        f = f.read()
        splited = f.splitlines()                        #split each line 1 on 1 into list

        splited = [i.split() for i in splited]          #split each element in arr to sub list in from [['asdasd', 'add', '1', '2', '3'], ...]
        # print(splited)                                #for debugging purposes

        instList = self.__simplify(splited)             #contains all assembly code in instruction list format like [labels, instcode, regA, regB, destReg]
        self.__assembly= instList
        # print(*instList,sep='\n')                     #for debugging purposes

        self.__fillFinding()
        # print(self.__fillValue)                       #for debugging purposes

        # self.translator(['None', 'sw', '7', '1', 'stack'])
        for element in instList:
            if(self.errorDetect == False):
                # print(element)                        #for debugging purposes
                self.translator(element)

            else:
                print("")
                print("exist(1)")
                print(self.errorDetail)
                print("")
                break

        if(self.errorDetect == False):
                print("")
                # print(self.__machineLang,sep='\n')        #for debugging purposes
                print("")
                print("exist(0)")
                print("")




if __name__ == "__main__":

    asbt = AssemblyTranslator()

    asbt.stringReader()

    asbt.printer()

    #then read instList and decode them as binary string
```