MIPS模拟机

3180103772 张溢弛

1.实验内容

以程序模拟MIPS运行，功能包括：
汇编器：将汇编程序转换成机器码。能有较灵活的格式，可以处理格式指令、表达式、有出错信息。
汇编反汇编：MIPS汇编指令与机器码的相互转换。
模拟器：根据机器码模拟执行可以运行简单MIPS程序。

1. 模拟器运行界面设计：可以命令行或窗口界面。可以执行指令的汇编、反汇编，可以单步执行指令观察寄存
   器、内存的变化。（命令行版可参考DEBUG）
2. 指令伪指令的汇编反汇编：将MIPS指令转换成二进制机器码，能够处理标号、变量。
3. MMU存储器管理单元：存储器存取模拟。大头小头，对齐不对齐，Cache，虚拟存储。
4. 格式指令表达式处理：对于汇编程序中的格式指令、表达式的处理。参考网页格式指令。

个人版模拟器应实现：

```
指令：R、LW、SW、BEQ、J五条；
命令：
-->R-看寄存器，
-->D-数据方式看内存，
-->U-指令方式看内存，
-->A-写汇编指令到内存，
-->T-单步执行内存中的指令
```

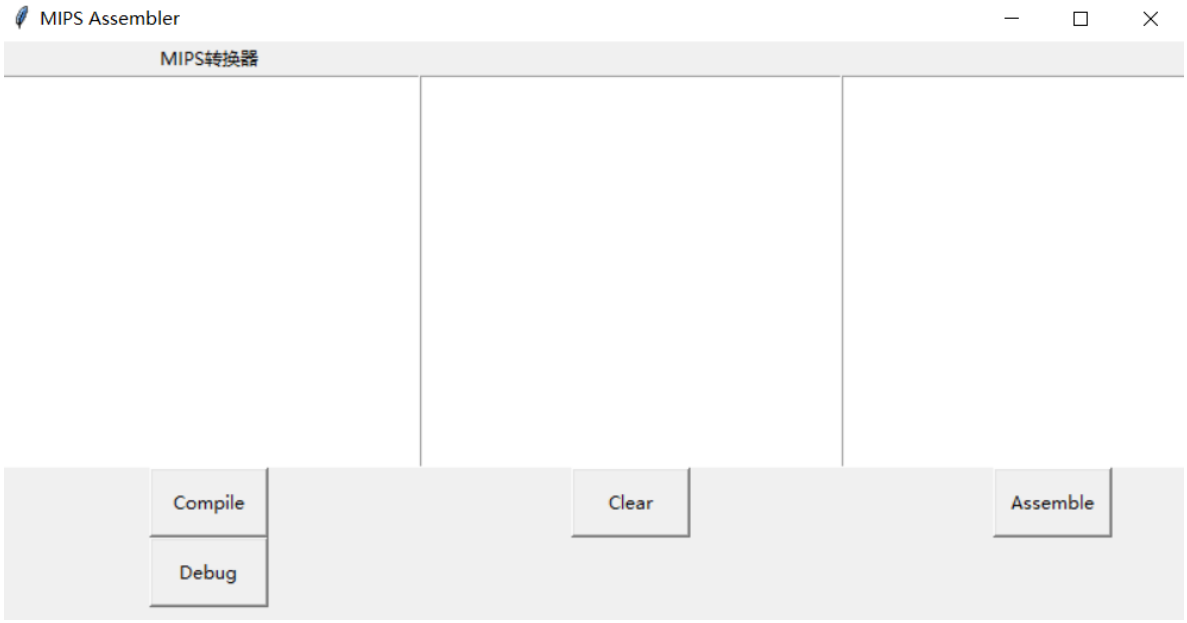
2. 实验环境

- 操作系统：Windows 10
- 语言： Python + PyQt 5 图形库
- 集成开发环境： Pycharm 2019.3 x64

3. 程序分析

3.1最终效果

- 采用了跨平台的Qt图形库制作界面，成品的结果如下



- 具体的功能包含
  - 机器码反汇编成MIPS指令
  - 执行反汇编后的MIPS指令
  - Debug模式，显示每一步指令的执行结果
  - Clear已经生成的MIPS指令

3.2 程序架构与算法分析

- 程序共分为3个py文件，分别实现了总体架构中的三个子功能
  - 界面交互
  - 机器码的翻译
  - MIPS汇编指令的执行
- 界面架构
  - 采用PyQt5图形库绘制界面，通过多个button按钮实现前后端的交互：包括指令的翻译，执行和调试
- 核心类1：机器码转换类 `class MIPS`，包含如下内容
  - 存储RIJ三类指令对应的机器码的字典

```
# the basic instructions of mips
# the R instructions
instruction_r = {"100000": "add", "100001": "addu", "100010": "sub",
"100011": "subu", "100100": "and","100101": "or", "100110": "xor",
"100111": "nor", "101010": "slt", "101011": "sltu","000000": "sll",
"000010": "srl", "000011": "sra", "000100": "sllv", "000110": "srlv",
"000111": "srav", "001000": "jr"}
# the I instructions
instruction_i = {"001000": "addi", "001001": "addiu", "001100": "andi",
"001101": "ori", "001110": "xori","000100": "beq", "000101": "bne",
"001010": "slti", "001111": "lui", "100011": "lw","100001": "lh",
"100101": "lhu", "101011": "sw", "101001": "sh"}
# the J instructions
instruction_j = {"000010": "j", "000011": "jr"}
```

  - 存储32个寄存器对应机器码的字典

```
register = {"00000": "$zero", "000001": "$at", "00010": "$v0", "00011":
"$v1", "00100": "$a0", "00101": "$a1","00110": "$a2", "00111": "$a3",
"01000": "$t0", "01001": "$t1", "01010": "$t2", "01011": "$t3","01100":
"$t4", "01101": "$t5", "$01110": "$t6", "01111": "$t7", "10000": "$s0",
"10001": "$s1","10010": "$s2", "10011": "$s3", "10100": "$s4", "10101":
"$s5", "10110": "$s6", "10111": "$s7",
"11000": "$t8", "11001": "$t9", "11010": "$k0", "11011": "$k1", "11100":
"gp", "11101": "sp","11110": "fp", "11111": "ra"}
```

- 用于翻译指令的各个成员函数

```
def getInstruction(self, op):
def getRegister(self, r):
def getShamt(self):
def translate(self):
def translate_R(self):
def translate_I(self):
def translate_J(self):
def getImmediate(self, n):
```

- 实现机器码转换成汇编指令的算法
  - 先读取操作码进行指令的基本分类，转换到对应的指令转换函数中进行进一步的翻译
  - 按照每种指令的特点进行操作寄存器和立即数，shamt，function code等内容的读取，通过字典转换成对应的MIPS汇编
  - 保存一份MIPS汇编指令作为模拟执行时使用，再通过join方法合并成符合语法规范的MIPS指令传递到前端进行显示
  - 具体的代码实现可以在上述成员函数中看到

- 核心类2： 用于执行和调试指令的类`class Assembly`
  - 32个寄存器的初始值

```
register = {"$zero": 'NULL', "$at": "NULL", "$v0": 'NULL', "$v1":
'NULL', "$a0": 'NULL', "$a1": 'NULL',"$a2": 'NULL', "$a3": 'NULL',
"$t0": 'NULL', "$t1": 'NULL', "$t2": 'NULL', "$t3": 'NULL',"$t4":
'NULL', "$t5": 'NULL', "$t6": 'NULL', "$t7": 'NULL', "$s0": 'NULL',
"$s1": 'NULL',"$s2": 'NULL', "$s3": 'NULL', "$s4": 'NULL', "$s5":
'NULL', "$s6": 'NULL', "$s7": 'NULL',"$t8": 'NULL', "$t9": 'NULL',
"$k0": 'NULL', "$k1": 'NULL', "gp": 'NULL', "sp": 'NULL',"fp": 'NULL',
"ra": 'NULL'}
```

  - 指令的模拟执行函数

```
def execute(self, instructions):
    i = 0
    # print(len(instructions))
    while i != len(instructions):
        i = self.singleLine(instructions[i], i)
    return self.getResult()
```

  - 指令的debug模拟函数

```
def debug(self, instructions):
    i = 0
    # print(len(instructions))
    result = ""
    while i != len(instructions):
        result += "Execute Instruction " + str(i+1) + ": " + "
".join(instructions[i]) + "\n"
        i = self.singleLine(instructions[i], i)
        single_step = self.getResult()
        result += single_step
     return result
```

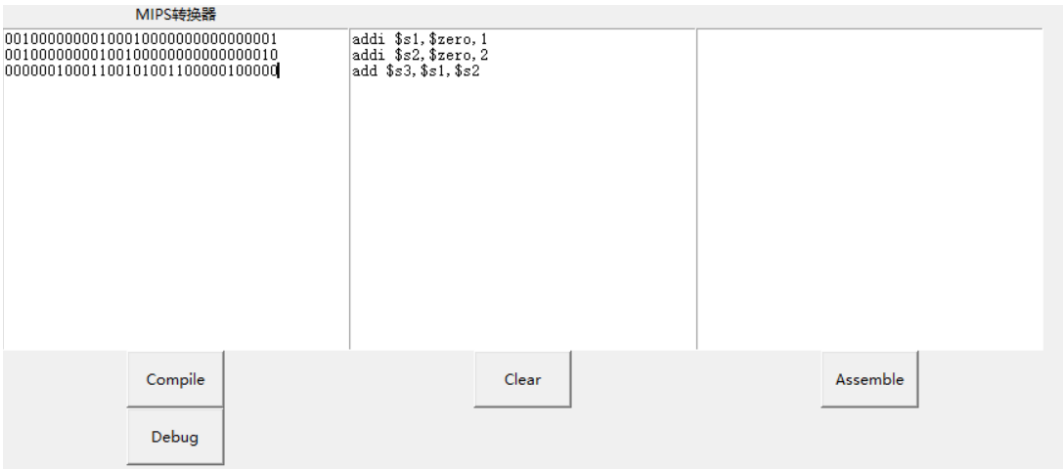
- 每条指令对一个单独的处理函数，用于对寄存器中的值进行改变
- 支持数组的地址类读取，Python是动态类型语言，lw函数在一定情况下可以将寄存器指向的内容为一个数组，从而实现通过数组的下标来寻址
- 执行过程的逻辑是：每次执行一条指令，函数返回的结果是下一条指令所在的地址，如果是普通的R指令和I指令，则返回下一条指令的地址，如果是跳转类的指令则会返回需要跳转到的行号
- **算法的实现方式**
  - 根据对应的指令，执行对应的指令函数，对存储在字典中的寄存器的值进行运算
  - 待执行的指令以列表的形式存储并传递给指令函数，不需要处理逗号空格等处理，因为是在机器码转换过程中直接传递给执行部分的
  - 每次执行一条指令，函数返回的结果是下一条指令所在的地址，如果是普通的R指令和I指令，则返回下一条指令的地址，如果是跳转类的指令则会返回需要跳转到的行号

3.3 支持的指令类型

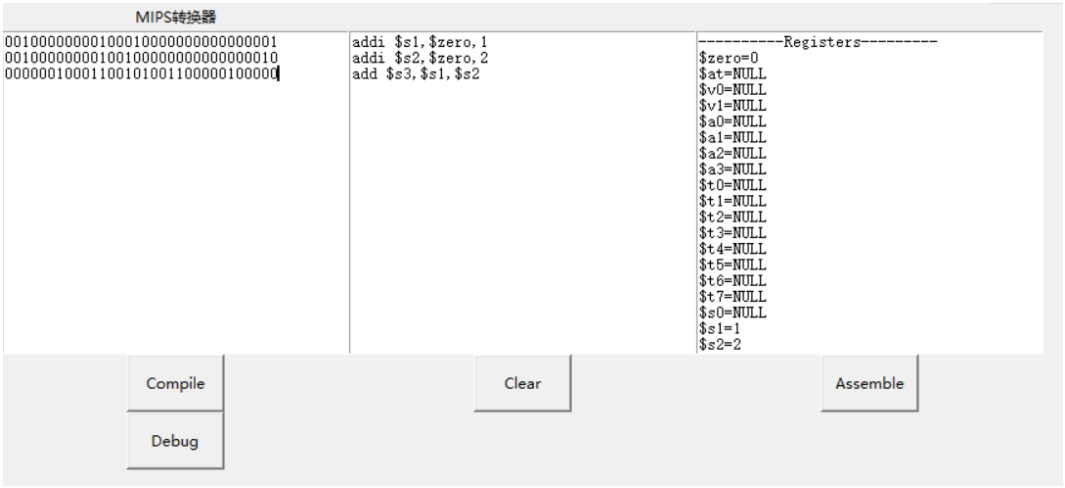
- 本程序支持基础的32条MIPS汇编指令包含
  - 全部R指令
  - 全部基本的I型指令
  - J型指令

4. 程序测试

1. **普通指令的测试(3条)**
   - 输入几条由add和addi组成的指令
   - 先点击compile生成对应的MIPS汇编指令

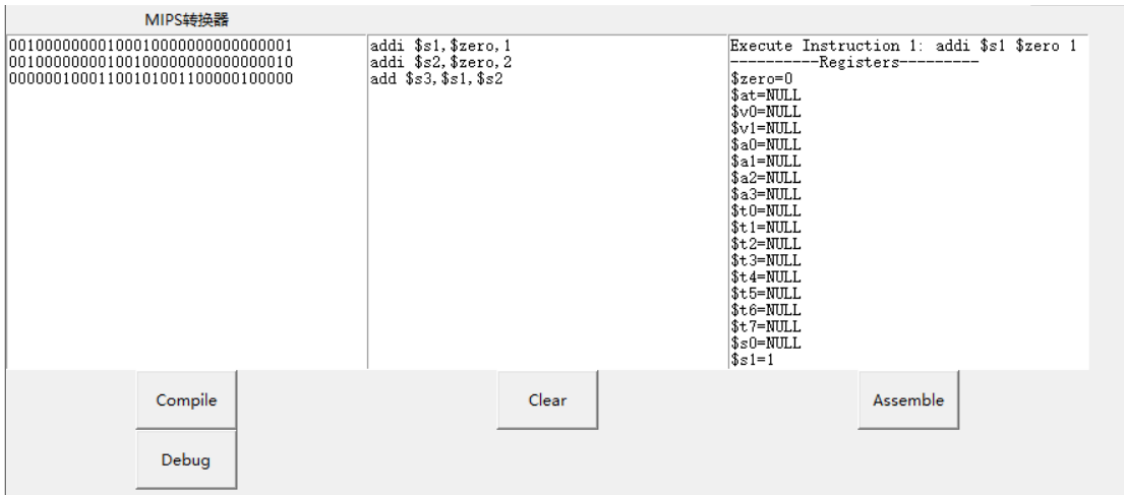


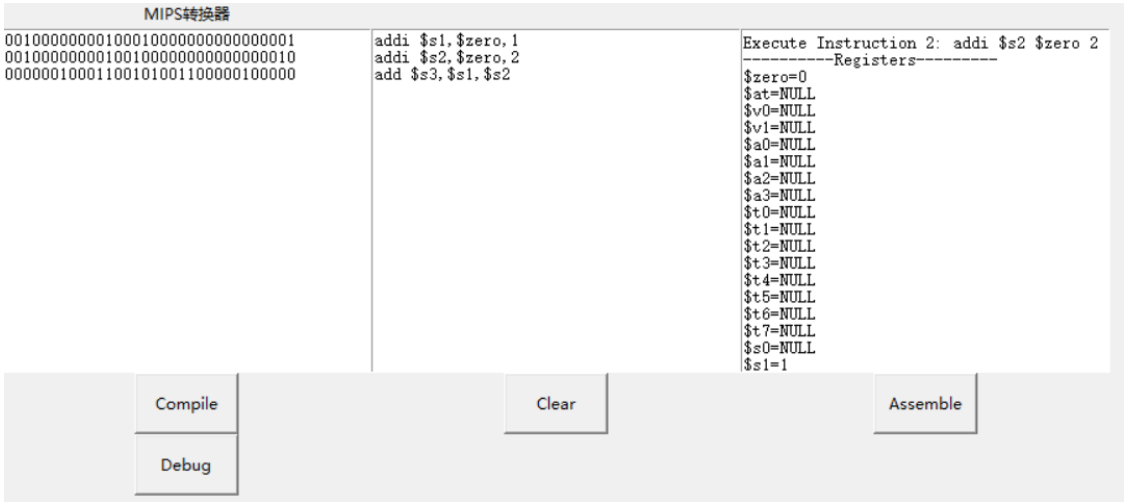
   - 再点击Assemble查看运行三条指令后的结果，可以查看到部分寄存器中的值发生了改变，指令被成功执行了



2. **Debug模式测试**

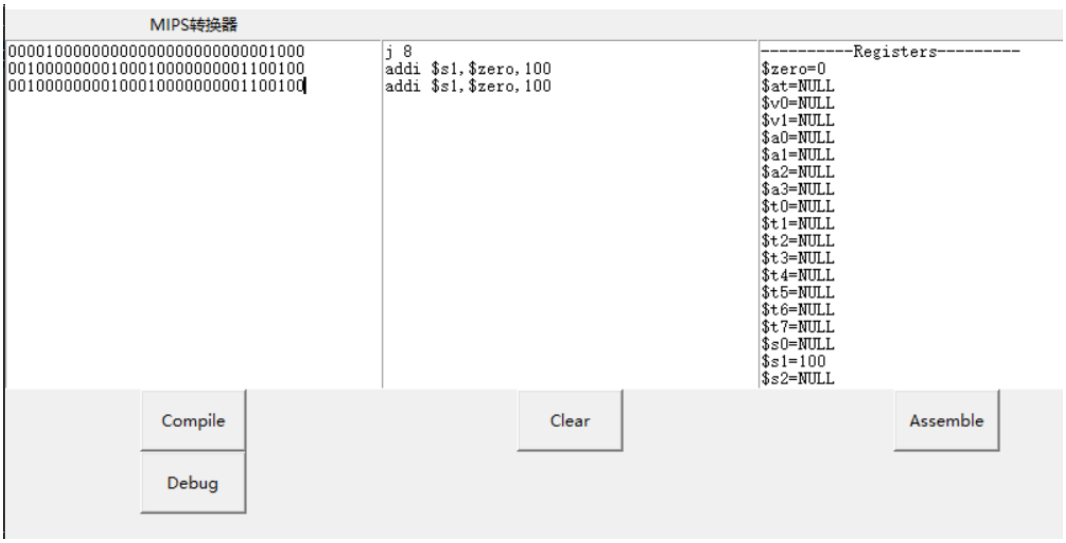
   - 同样先来输入三条指令并进行反汇编
   - 点击Debug按钮，进行debug，第三个文本框中会显示出执行指令的过程和每次指令执行之后32个寄存器中的对应的值，可以让用户进行debug





3. **特殊指令的执行测试**

   - J类指令的测试
     - 这里测试直接跳转到第二条指令执行，测试结果是成功的



- bne指令的测试
  - 这里两个数不相等，因此应该跳到第四行执行(只能加一次100)，最后寄存器显示的结果也是1次，因此bne指令的执行是正确的