计算机系统原理实验报告

课程名称：计算机系统原理 实验类型：上机

实验项目名称：MIPS 汇编

学生姓名：应承峻 专业：软件工程 学号：3170103456

实验日期：2019 年 3 月 23 日

一、实验描述

用 C 或 JAVA 编写程序，将 MIPS 汇编指令转换为机器码，并以二进制方式存盘，以作为随后反汇编的输入及 CPU 模拟器的执行程序。

实际汇编指令到机器码的转换。可以选择多种输入输出方式，可以是单条指令即输即显。也可以是选写好汇编源程序文件，以文件读入二进制文件输出。说明文档：所实现的指令、程序框图、使用方法、实例分析等等。

- 采用尽可能多的指令。
- 考虑伪指令。
- 考虑符号[标号、变量]
- 考虑表达式[如：SW $s1, 4*12($s0)]

二、程序实现的指令

本实验主要实现的指令如下（所有指令支持大小写）：

- R 类型指令
  - add rd, rs, rt
  - sub rd, rs, rt
  - slt rd, rs, rt
- I 类型指令（其中 expr 为算数表达式，label 为跳转到指令响应的标签）
  - lw rt, expr(rs)
  - sw rt, expr(rs)
  - addi rt, rs, expr
  - beq rs, rt, label
  - bne rs, rt, label

- J 类型指令
  - `j label`
- 伪指令
  - `move r1, r2`
  - `blt r1, r2, label`
  - `bgt r1, r2, label`
  - `ble r1, r2, label`
  - `bge r1, r2, label`

三、程序设计思路（本实验采用的语言为 Java）

- 输入形式与输出形式

  在本实验中，指令应存储于根目录下'***in.txt***'文件中，每条指令占一行，允许空行存在，最后以占单独一行的 ***end*** 标记结束。输出时，控制台按照固定格式每行依次输出指令地址、十六进制指令以及反汇编结果，并将二进制指令存盘于根目录下的'***out.txt***'文件中。

- 指令的预处理流程及代码实现
  - 空行处理：因指令允许空行存在，故若去掉指令中其中的\t 转义字符以及空格后指令为空，则直接读取下一行指令，其代码可表示为：

    ```
    while (instruction.replace('\t', ' ').trim().equals(""))
            instruction = br.readLine();
    ```

  - 标签处理：先通过正则表达式匹配 ***label:***形式的字符串，然后将其从字符串中去除，以便之后对指令的处理，该字符串可用如下正则表达式匹配：

    `^([a-z 0-9]+):`

    处理完标签后将标签、指令以及计算好的指令地址以 Instructions 类的形式存储到指令容器 ArrayList<Instructions>中。

  - 指令处理：由于程序允许使用大小写指令，因此首先需要把字符串全部转换成小写形式，然后将逗号，转义字符\t 全部替换成普通空格，最后通过正则表达式使用 split 函数将其分割。
- 指令汇编与反汇编
  - R 类型指令处理：R 类型指令其通用格式为

op	rs	rt	rd	shamt	funct
6	5	5	5	5	6

因而可通过以下方式处理指令的各个部分，以 ***add rd,rs,rt*** 为例：

```
split 函数分割后得到 is = {"add","rd","rs","rt"}
instruction = (reg_index(is[2]) << 21) +   //rs
(reg_index(is[3]) << 16) +          //rt
(reg_index(is[1]) << 11)     //rd
+ 32       //funct: 100000
```

- I 类型指令处理：I 类型指令其通用格式为：

op	rs	rt	immediate
6	5	5	16

对 op,rs,rt 的处理与 R 类型指令的处理方式类似，而当 immediate 处的值为表达式时，需要通过 Java 自带的引擎 ***ScriptEngineManager*** 来进行表达式的计算，计算完毕后得到立即数后再进行处理。当指令为 beq 或是 bne 时，immediate 处是一个 label，此时需要从预存标签的 instructions 类的容器中寻找到该标签所处的地址，再计算 PC 值之差即可得到 immediate。

此处以 ***beq rs rt label*** 为例：

```
delta = label_pc(is[3]) - (instruction.pc + 4); //获取 delta
instruction = (4 << 26) +   //op code
(reg_index(is[1]) << 21) +   //rs
(reg_index(is[2]) << 16) +   //rt
((delta >> 2) & 0xffff)      //immediate 需要右移两位
```

- J 类型指令处理：J 类型指令的通用格式为

op	address
6	26

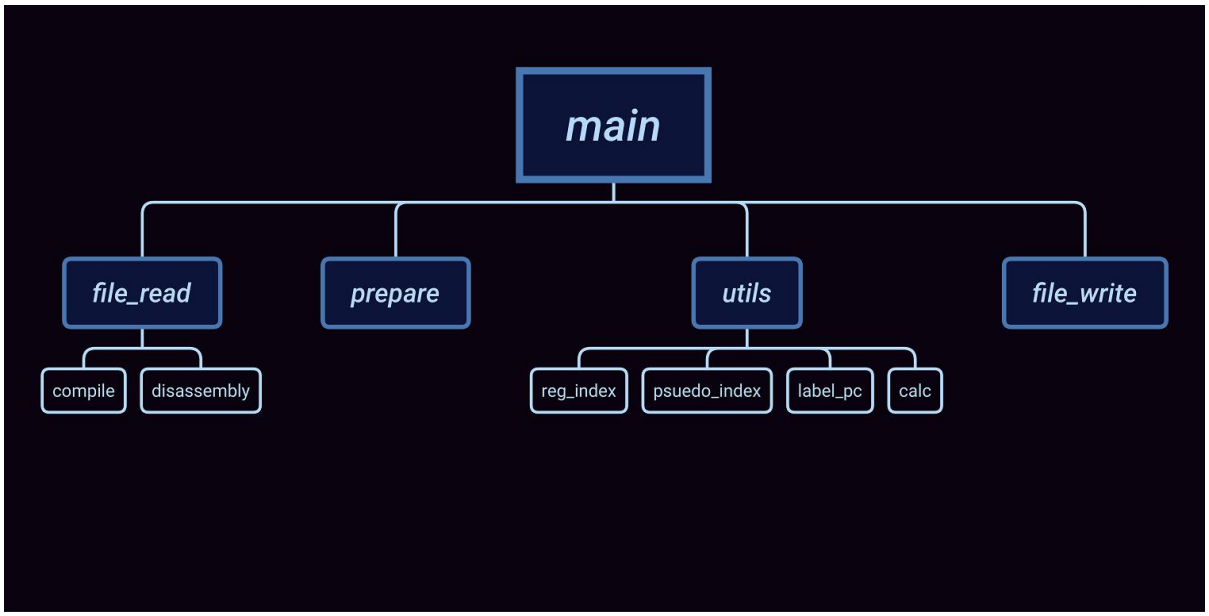
对 op 的处理类似于 R 类型指令的处理，对 address 中的标签的处理类似于 I 类型指令中 beq 和 bne 指令的标签的处理，因而在这以 ***j label*** 为例：

```
delta = label_pc(is[1]) - (instruction.pc + 4);
instruction = (2 << 26) +   //op code
```

```
((delta >> 2) & 0x3ffffff))   //address
```

- 伪指令处理：对伪指令的处理与普通指令的处理相同，只需将其按照普通逻辑的指令进行分条处理即可。但伪指令需要在输出处理上做一些准备。

四、程序框图



五、实例分析

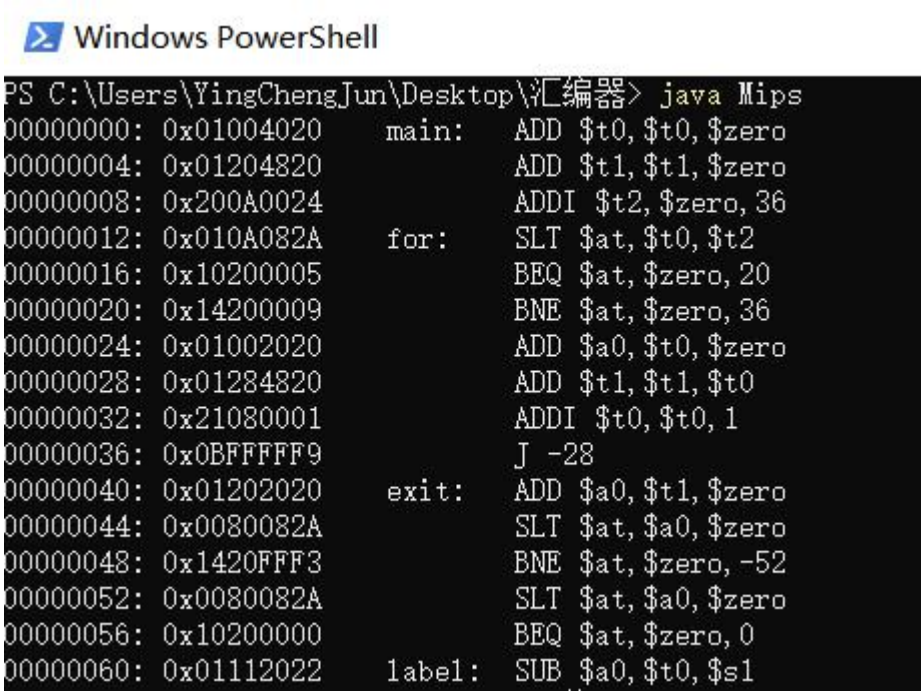
在 ***in.txt*** 中存放了如下指令：

in.txt - 记事本

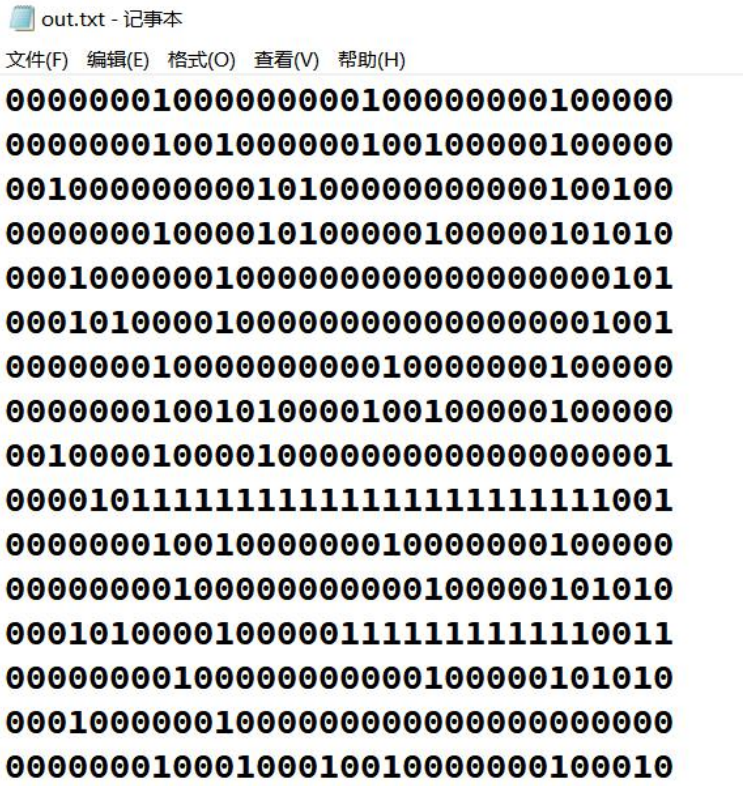
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
main:   add     $t0,  $t0,   $zero
        add     $t1,  $t1,   $zero
        addi    $t2,  $zero, 36
for:    slt     $at,  $t0,   $t2
        beq     $at,  $zero, exit
        bne     $at,  $zero, label
        move    $a0,  $t0
        add     $t1,  $t1,   $t0
        addi    $t0,  $t0,   1
        j       for
exit:   move    $a0,  $t1
        blt     $a0,  $zero, main
        bge     $$a0, $zero, label
label:  sub     $a0,  $t0,   $s1
end
```

为了确认指令汇编的正确性，我们通过反汇编指令来验证。执行该程序后，控制台输出：



程序在 ***out.txt*** 中输出：



实验结果与预期结果相符合。

六、实验心得

在本次实验中，个人认为比较棘手的问题就是标签的处理，为此，我建立了一个

Instructions 类，该类中含有三个变量：int 类型的 pc 用于存储指令地址，String 类型的 label 用于存储标签，String 类型的 content 用于存储指令内容，在预处理完后，将指令以类的形式存储可以便于后续的处理。通过这次实验，我对基本的 R 类型、I 类型和 J 类型指令以及伪指令有了更深入的理解，使得对指令及其寄存器的用途更加熟悉。