实验二 选题一 Tomasulo算法实现

杨骐瑞 计96 2019011347

实验环境

OS: windows 10

Python: 3.10.5 **32 bit**

g++: clang version 14.0.0 Target: x86_64-w64-windows-gnu

文件架构

Tomasulo

|__ convert

| |__ input1-converted.txt & other machine code file

| ……

|__ sample

| |__ input1.asm & other asm file

| ……

|__ api.py & other source code

……

实验主要框架描述

由于实现了 gui，本项目分为两个分支仓库地址，其中 master 分支为无 gui 版本，gui 分支为对应有 gui 的版本

使用前，需要先使用根目录下的converter.py脚本将所需运行的 asm 文件转换成机器码

converter.py 使用示例：

```
python3 converter.py -file input1.asm
```

默认输出文件名为 ${source file name}-converted.txt 。然后将template.cpp进行编译，并将机器码文件输入即可。

- gui 分支

  template.cpp 仅将 main 函数拆分为了初始化 init 函数和 run_one_tick 单步执行函数，其余几乎未作改动。

  gui 分支下，图形界面采用 tkinter 完成。此外，为了直接在 python 中调用 c 程序，采用了 python ctypes 库，并将 template.cpp 中所需用到的函数和定义进行 extern 并用 g++ 进行链接为 dll 方便后续在 python 中直接使用，这一步避免了使用文件流进行通信和复杂的序列化反序列化。具体编译命令如下：

  ```
  g++ -o .\template.dll -shared -static -m32 -fPIC .\template.cpp
  ```

注意到这里采用的是**32 bit** 的编译命令，所以 python 版本也需要使用**32 bit**，否则使用时会提示为不是正确的 win32 文件。

api.py 是利用 ctypes 完成的 python 和 cpp 通信接口，其中类的定义如下例：

```
class machineState(Structure):
    _fields_ = [
        ("pc", c_int),
        ("cycles", c_int),
        ("reservation", resStation*RES_STATION_SIZE),
        ("reorderBuf", reorderEntry*ROB_SIZE),
        ("regResult", regResultEntry*REG_SIZE),
        ("btFuf", btbEntry*BTB_SIZE),
        ("memory", c_int*MEM_SIZE),
        ("regFile", c_int*REG_SIZE),
        ("headRB", c_int),
        ("tailRB", c_int),
        ("clear", c_int),
        ("memorySize", c_int),
        ("halt", c_int)
    ]
```

只需要将需要使用到的类继承 ctypes 中的 Structure 类，并将其各个变量设置为和 cpp 类类型完全一致，即可将 cpp 的执行结果自动解码为 python 类。例如：

```
template = ctypes.cdll.LoadLibrary("template.dll") # 导入dll
c_run_one_tick = template.run_one_tick # 定义单步执行函数
c_run_one_tick.argtypes = [machineState] # 定义函数参数类型
c_run_one_tick.restype = machineState # 定义函数返回值类型
```

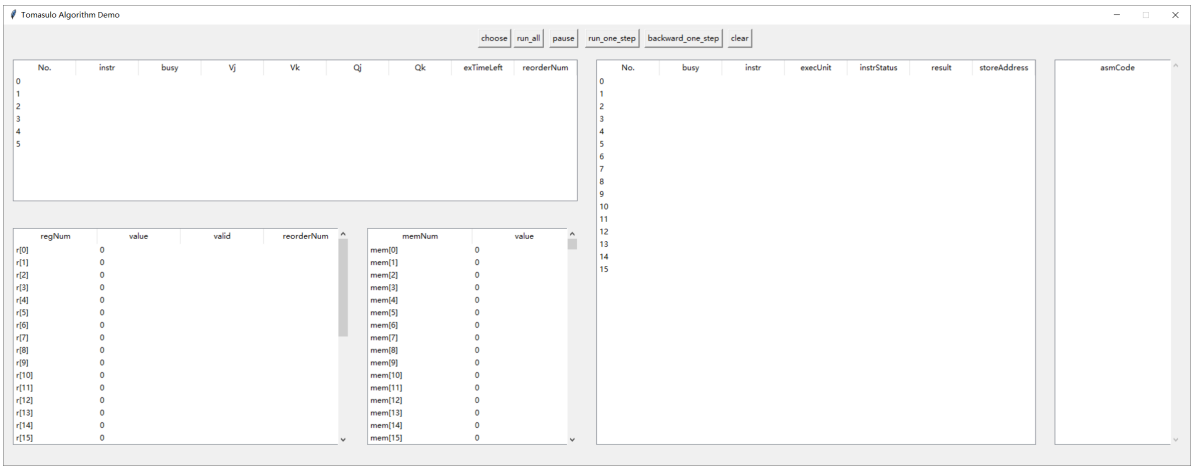
此时 c_run_one_tick 实际上就是一个 python 函数，它接受 machineState 为参数，并调用 cpp 编译的函数执行，再得到同样类型的返回值。

根据上述原理，为了能够让前端自由得到每一个时钟周期的状态机，使用列表存储下所有时刻的状态，并利用列表的 pop 和 append 方法实现前进后退。

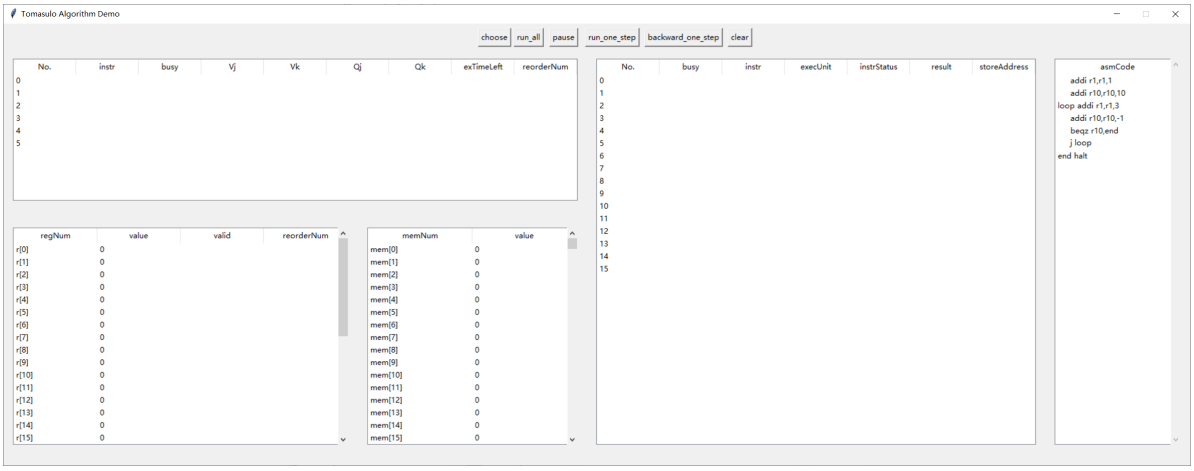
api.py 中实现了：获取最新状态 get_present，弹出最新状态（后退一步）states_pop，获取列表长度 states_len，初始化 init_states，单步执行 run_one_tick 五个简单操作。

gui.py 主要利用 Tkinter 库完成，其中画表部分采用 Treeview 控件，只需要每次获取状态机并将其中的内容填表即可，较为简单。

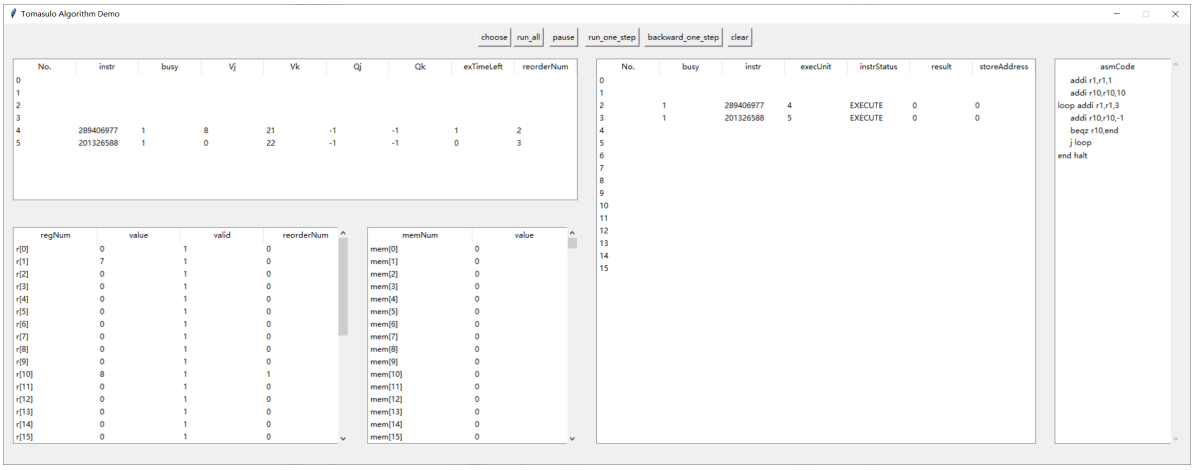
启动命令：`python gui.py`



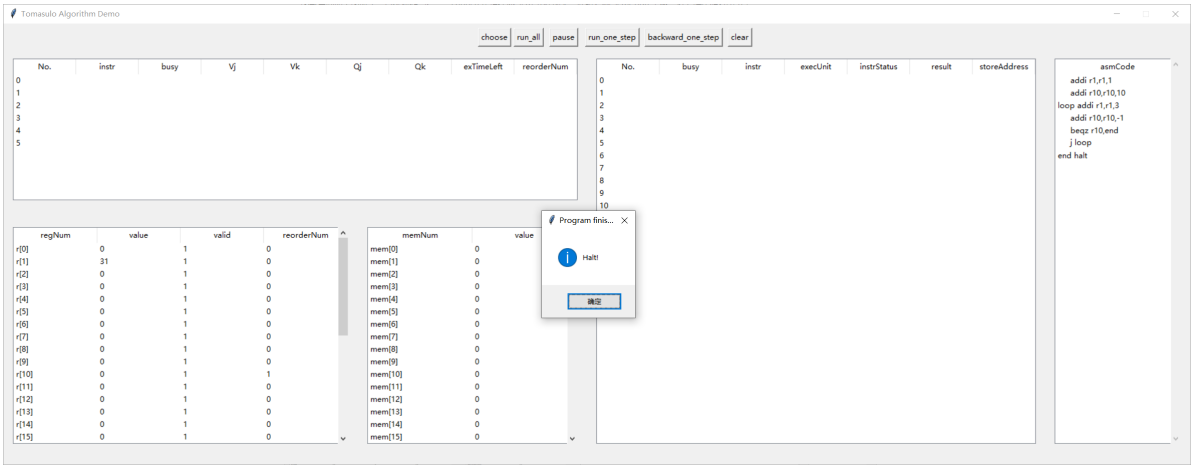
图形界面如上图所示，具体操作流程是，choose 按钮选择 asm 文件读入，此时右侧 asmCode 表格会将汇编代码打印出来：



run_all 按钮会让程序从当前状态一直运行到停止为止，pause 按钮则可以让这个过程停止。run_one_step 可以让当前状态运行一个周期，backward 按钮可　　以让当前状态回滚一个周期（请勿在 run_all 的过程中运行 run_one_step 或 backward，可能导致状态机出现问题，尽量在暂停状态下使用），clear 则让 g　　ui 程序恢复初始状态（也请在程序运行结束或暂停的清况下清空）



运行中的程序如下，各个部分的表格将会实时刷新。当任务完成后，将会提示停机：



实验实现细节：

- converter

  实现思路是，首先利用 split 函数将注释和多余空格去掉，然后将头部不属于操作符的标号的行号和对应名称计入字典，再次遍历时，如果尾部有字典中的标号，就计算出偏移值。然后进行按位与操作即可输出整数格式的机器码

- 虚拟机
  - 指令各个域解码：该部分只需要对指令进行位移和按位与操作即可，例如：

```
int field2(int instruction) {
  /*
   *[TODO]
   *返回指令的第三个寄存器，Rd
   */
  int reg3 = (instruction >> 11) & 0x1f;
  return reg3;
}
```

  - 保留栈更新：根据代码，Qj/Qk 表示某一保留栈项所等待的保留栈项标号。只需要遍历所有保留栈，并将 Qj/Qk 满足的项的 Vj/Vk 置为所对应的值即可。

```
void updateRes(int unit, machineState *statePtr, int value) {
  /*
   *[TODO]
   * 更新保留栈:
   * 将位于公共数据总线上的数据
   * 复制到正在等待它的其他保留栈中去
   */
  for (int i = 0; i < NUMUNITS; i++) {
    if (statePtr->reservation[i].Qj == unit) {
      statePtr->reservation[i].Vj = value;
      statePtr->reservation[i].Qj = -1;
    }
    if (statePtr->reservation[i].Qk == unit) {
      statePtr->reservation[i].Vk = value;
      statePtr->reservation[i].Qk = -1;
    }
  }
}
```

  - 发射指令：将对应得 ROB 项和保留栈项的相关项的各个域按照注释所述填写正确即可。

- 检查 ROB 空闲：考虑到本实验的计算延迟最多不超过3，功能部件最多不超过2个，所以只需检查 tailRB - headRB 是否等于 ROB 大小即可
- 计算结果：按照不同的指令计算即可。注意要根据位数进行转换

```
else if (op == LW) {
    return statePtr->memory[rStation.Vj + convertNum16(immed)];
}
```

- 主循环（单个时钟周期）：
  - COMMITTING：分两种情况考虑：跳转 or 非跳转。若需要跳转，则需要清空 ROB 和保留栈，并将头尾指针归零，寄存器恢复valid；若非跳转，则根据指令的不同，对寄存器或内存进行操作。**此处多了一个 updateRes，是因为如果在 Writing Result阶段紧接着 Issue 了一条指令，新发射的指令并不会采用写回的结果（实验框架决定），所以必须要再一次进行保留站更新从而杜绝这种问题**
  - 其余状态：
    - ISSUE：若两个寄存器均准备好（没有冲突），进入 EXECUTING 阶段
    - EXECUTING：减少剩余时间，如果时间等于0，调用计算结果函数获取计算结果并写入 ROB 对应项的 result 域
    - WRITING RESULT：将所有依赖结果的保留栈通过调用保留栈更新函数进行更新结果，并将对应使用结束的保留栈置为空闲，然后转入COMMITING 状态
  - 发射新指令：同时检查 ROB 和保留栈相关计算单元是否空闲，均空闲则发射新指令，并将 pc + 1。需要注意全0的指令在本实验框架中也作为合法指令，所以需要限定 pc 不超过 memorySize，否则会发射 add r0,r0,0 这样的指令
  - 周期数 + 1

备注

图形界面 tkinter 以及 ctypes 的选用与聂鼎宜、卢鹏交流得出，在此之外并未有代码的抄袭。图形界面的鲁棒性不高，并未对 button 等控件的 enable 状态进行限制，在各种反复点击和非法操作下可能会出错，敬请谅解。