高体第二次作业—Scoreboard和Tomasulo算法描述

一. Scoreboard算法

1. 简介

我们组基于C++语言实现了一个记分牌算法的描述程序，该程序支持LOAD、ADD、SUB、MULT和DIV五种指令。输入是一个指令集合，输出是指令状态、计分板状态和目的寄存器状态三个状态表。

我们还提供了一个pdf用来检测运行结果的正确性，该pdf名字为dynamic.pdf，可以对照其来检测Scoreboard和Tomasulo每一步状态表的正确性。

**代码文件结构：**

----Scoreboard

|----variables.h

|----variables.cpp

|----main.cpp

**编译环境和方法：**

本程序在VS2015下进行编写并测试运行成功。经测试，本程序也可以运行在VS2013和Codeblocks下。不过低于VS2012版本不能顺利运行该程序，请VS2012的用户使用新的VS进行编译运行。

在VS2015和VS2013下，创建新的WIN32控制台项目，在选项中将预编译头和安全开发周期两个选项取消，然后选中空项目选项，完成项目创建，然后正确引入一个头文件和两个代码文件，编译运行即可。

在Codeblocks下，创建新的项目，在该项目下引入所有文件，然后就可以编译运行。

2. 运行与测试

程序的入口在main.cpp文件中的main函数处。

程序提供两种指令输入模式：单指令输入模式和多指令输入模式。

**单指令输入模式：**

这个模式下一次输入一个指令，直到将要测试的指令集中的所有指令输入完毕，程序会自动统计指令总数。使用该模式需要将main函数中的指令输入注释下一行的int count的右值改为input\_inst()函数，编译运行即可。

接下来介绍所使用的测试指令集，这个指令集就是书上的例程，粘贴如下：

LD F6 0 R2

LD F2 0 R3

MULTD F1 F2 F4

SUBD F8 F6 F2

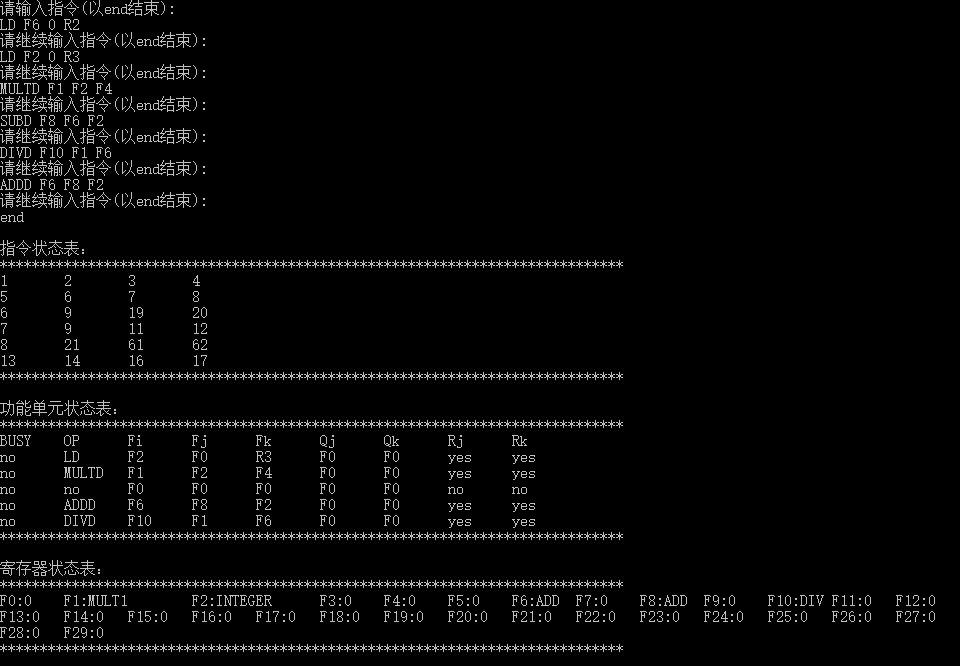
DIVD F10 F1 F6

ADDD F6 F8 F2

在这里做一下说明--1.和书上一样，定义的寄存器有F1-F10和R1-R4，对于R系列寄存器，指令可以直接读取，不考虑冲突和相关性，而F系列的寄存器要考虑相关性;2.对于目的寄存器状态表，因为**本程序默认的输出是所有执行完后的状态表**，所以在**每条指令执行完成后不将对应的目的寄存器状态表项清零**，而是保留，当然也可以选择清除该表项目(在execution()函数的while循环中，有个注释，执行完指令的目的寄存器表项清零，将注释下一行的注释符号去掉即可)

首先我们输入指令，然后运行，得到的结果如图1所示。指令状态表中每一个值是该阶段完成的周期数；功能单元状态表已经说得很详细了，这里只说如果哪一条中的OP项的值为no，说明该条功能单元没有指令使用过。该状态表和目的寄存器状态表一样，**每次指令执行完只将BUSY位置0，而不删除其他位置的值，而且读取完操作数之后也不将R字段置0，而是使其一直保持为yes，**这样便于观察；寄存器状态表不再介绍。

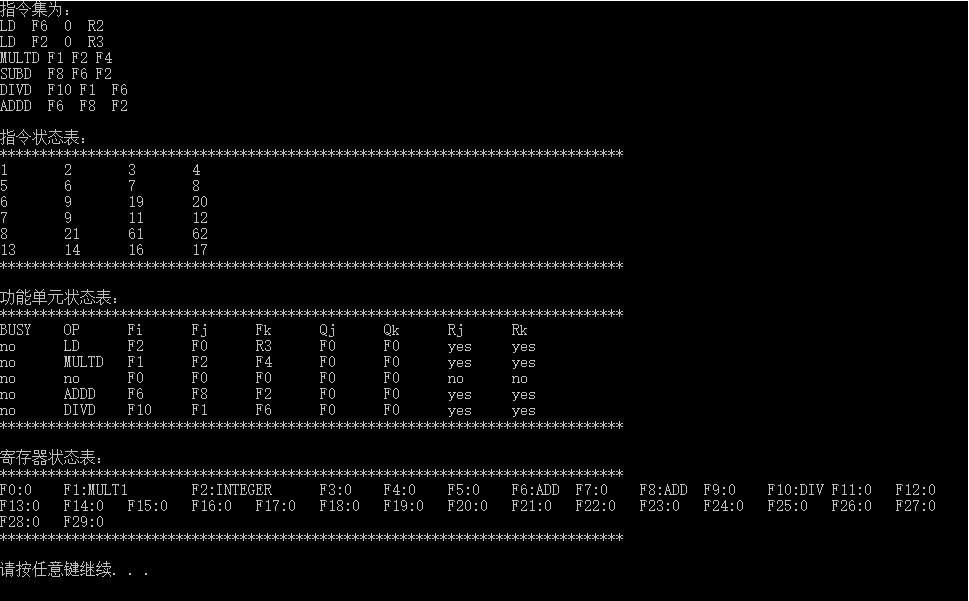
另外，**也可以让程序输出任意一个周期的状态表，只需要在main函数中的execution(count)函数中count参数后加入参数cycle，**即可查看cycle周期的状态表。



**图1**

**多指令输入模式：**

只需要将main函数中的指令输入注释下一行的int count的右值改为input\_inst\_exa()函数，编译运行即可。其他内容与单指令模式相同。如图2.



**图2**

如果要**自己编写指令集合**，那么只需要在input\_inst\_exa()函数中的instr数组中编写指令集合，然后根据指令的个数修改count常量的值。

3. 代码说明

variables.h：

这是声明变量的头文件，首先定义了五个枚举变量分别表示寄存器、指令类型、指令执行阶段、功能单元和功能单元状态表字段；然后声明了功能单元状态表、寄存器状态表和周期的外部变量；声明了指令的结构体，基于指令的结构体声明了指令集合类。

variables.cpp：

这个代码文件定义了指令集合类中的函数。

main.cpp：

这个代码文件中给出了Scoreboard算法的具体实现。首先要初始化状态表，然后输入指令集合，然后执行，最后输出状态表。

**关键的地方在执行部分**，我们采用了时间驱动的方式，就是每次循环周期数都会加1。每个循环中，都会在该周期内遍历指令集合中的所有指令，依次检查每条**指令处于哪个阶段**以及**是否可以进入下一个阶段**。判断指令处于哪个阶段是这样做得，首先在指令集合类中为每条指令设定了四个变量用来表示四个阶段开始的周期数，通过对每个阶段这个值是否为0来判断。判断是否可以进入下一个阶段是通过四个函数进行的，第一个函数判断函数是否可以发射，先检查RAW冒险，然后检查结构冒险，然后检查是不是按序发射(与前条指令的关系)，最后检查WAW冒险，都没有问题的话就发射，并修改状态表；第二个函数判断是否读取操作数，主要检查RAW冒险；第三个函数执行；第四个函数判断是否可以写回，检查WAR冒险，然后写会并修改状态表。每条指令都遍历之后，**寻找是否有写会的指令**，找到后对相关状态位进行归位。最后如果所有的指令都写回了，那么程序执行完毕。

二. Tomasulo算法

1. 简介

我们组基于C++语言实现了一个Tomasulo算法的描述程序，该程序支持LOAD、ADD、SUB、MULT和DIV五种指令。输入是一个指令集合，输出是指令状态、功能单元状态和目的寄存器状态三个状态表。

**代码文件结构：**

----Tomasulo

|----variables.h

|----variables.cpp

|----main.cpp

**编译环境和方法：**

同Scoreboard的编译环境和方法。

2. 运行与测试

本部分与Scoreboard的运行与测试部分基本相同，这里只说一下不一样的地方。

首先指令集合变为：

LD F6 34 R2

LD F2 45 R3

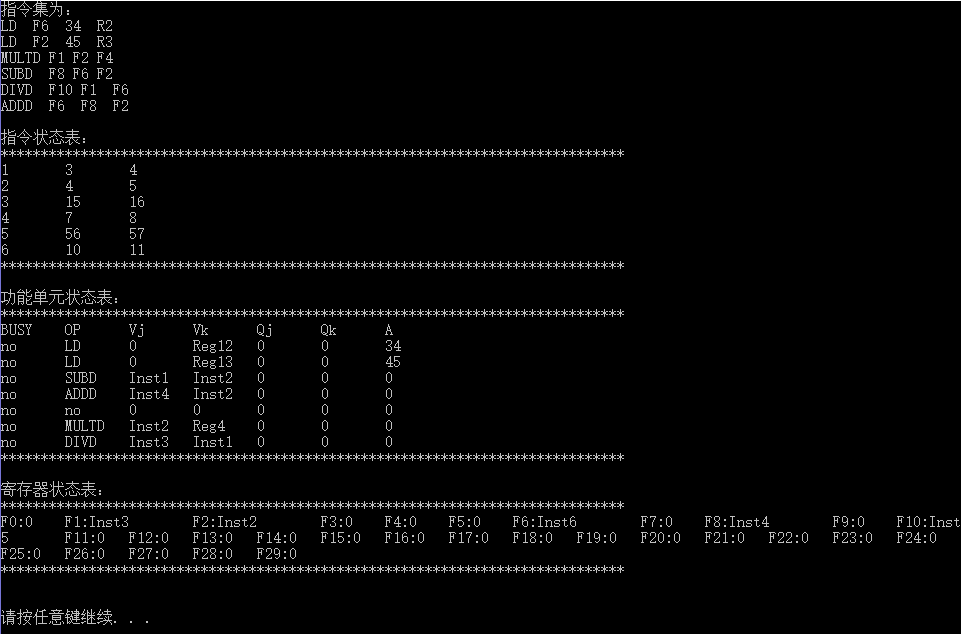
MULTD F1 F2 F4

SUBD F8 F6 F2

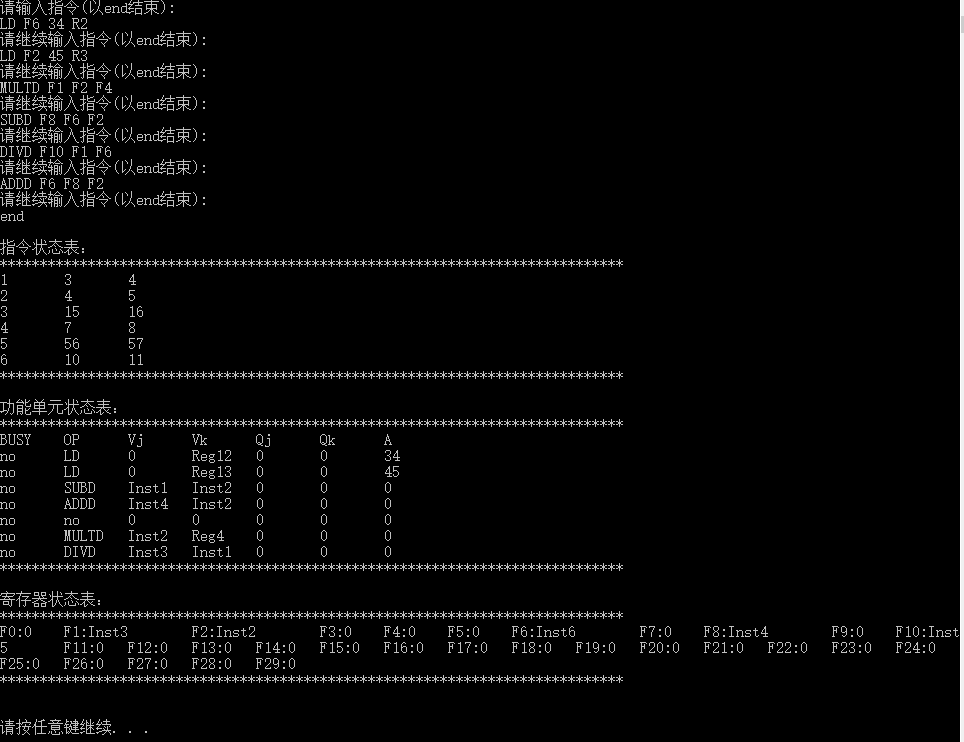
DIVD F10 F1 F6

ADDD F6 F8 F2

在单指令模式下运行得到结果如图3，多指令模式下得到结果如图4。



**图3**



**图4**

这里做两点特别的说明。1. **功能单元表中V字段的值是保留站的编号**，Inst i表示该保留站保存第i条指令产生的值，Reg i表示该保留站保存第i个寄存器的值，若i为11-14则说明这个寄存器是R1-R4。2. **寄存器状态表中每个寄存器对应的值有两种**，第一种则是表中所见的保留站编号，如果指令执行完成，那么就保存保留站编号；第二种是功能单元，如果指令正在执行，则保存运行指令的功能单元。

3. 代码说明

variables.h：

这是声明变量的头文件，首先定义了五个枚举变量分别表示寄存器、指令类型、指令执行阶段、功能单元和功能单元状态表字段；然后声明了保留站、功能单元状态表、寄存器状态表和周期的外部变量；声明了指令的结构体，基于指令的结构体声明了指令集合类。

variables.cpp：

这个代码文件定义了指令集合类中的函数。

main.cpp：

这个代码文件中给出了Tomasulo算法的具体实现。首先要初始化状态表，然后输入指令集合，然后执行，最后输出状态表。

**关键的地方在执行部分**，具体的执行流程和Scoreboard差不多，这里只说一些不同之处。首先判断发射的函数不需要检测RAW冒险和WAW冒险，发射的时候，如果生成源操作数的指令未执行完，则设置Q字段来等待该指令；如果已经执行完并写入保留站，则设置V字段为相应保留站的值；否则直接读取该寄存器的值，并将设置保留站来保存这个值，这里重命名操作解决了WAW和WAR冒险。然后判断读取操作数函数等待Q字段的指令，此步解决RAW冒险。然后再execution函数的循环体内，对于**执行完的指令，对其exc和相应功能单元表中的Busy位进行归位，其次还需要将为结果设置一个保留站来保存，然后对该指令之后的所有需要使用该结果的指令进行广播，将其Q字段置0，V字段设置成这个保留站编号**。其他的和Scoreboard差不多，此处不再多说。