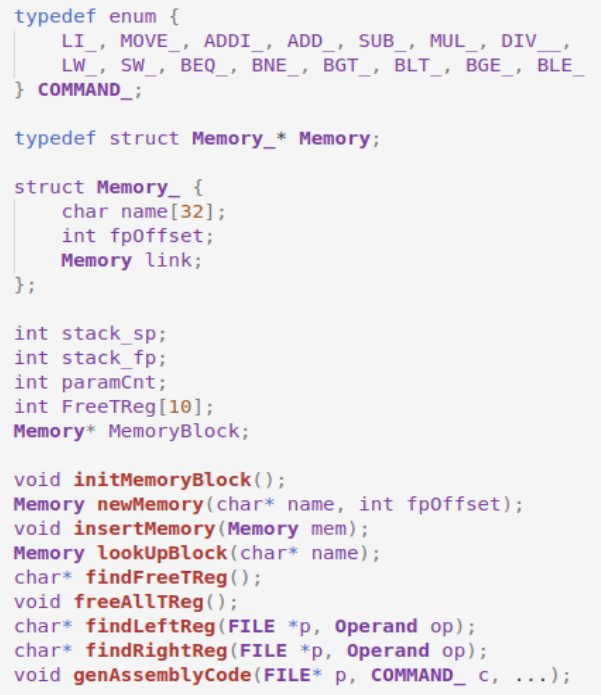
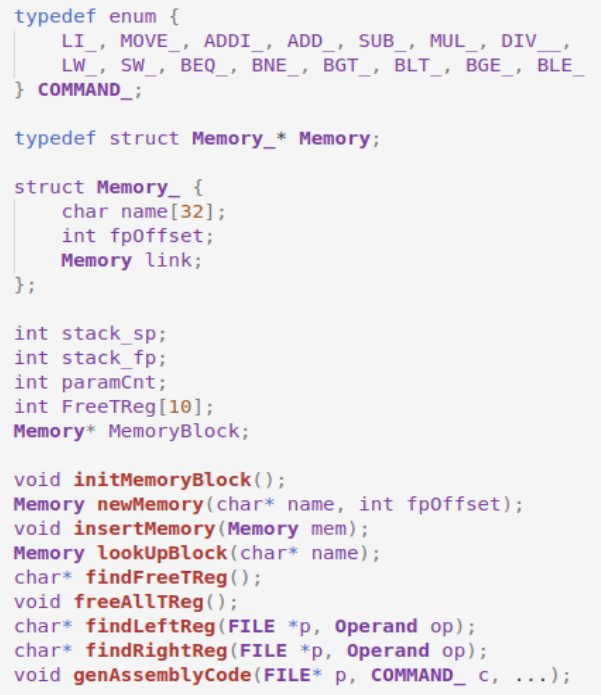
实验4 191220154 张涵之

1. 实现功能
   1. 必做内容：实现目标代码的生成；
2. 实现思路
   1. 在实验3基础上继续扩展lib.h和lib.c中定义的数据结构：

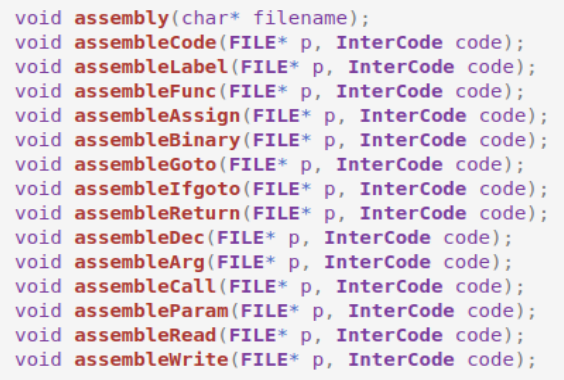


其中枚举类型COMMAND对应MIPS32中用到的指令，Memory Block及其相关函数用到了哈希表，类似实验2的Symbol Table。由于实验使用最简单的“朴素寄存器分配算法”，因此需要记录中间代码中变量和临时变量在栈中的偏移量。寄存器的分配和释放用数组FreeTReg来记录，整个实验基本只用到了$v0、$sp、$fp和由$t0-$t9表示的临时寄存器，其他内容都直接在栈上/内存中进行实现。



定义了Memory Block的初始化、插入和查找函数，findFreeTReg和freeAllTReg用于在FreeTReg数组中寻找空闲的临时寄存器和将全部临时寄存器设为空闲。另有函数findLeftReg和findRightReg，分别结合左值/右值操作数的具体类型，写指令到文件以实现取出栈中的相应变量，存放到一个空闲的临时寄存器中，并返回该临时寄存器的名称。genAssemblyCode根据调用MIPS32指令的类型读取参数（寄存器和/或立即数）并生成对应的目标代码，写入到文件中。

* 1. 继续定义了assembly.h和assembly.c模块，结合中间代码生成目标代码：



这里多亏实验3没有直接一边转化一边输出到文件，而是先建好中间代码的链表结构在逐条便利输出。函数assembly调用initMemoryBlock对内存内容进行初始化，打开对应文件写入部分通用data和text信息，如样例中所示的\_prompt、\_ret和read write函数等内容。接着遍历中间代码链表逐条调用assembleCode，这个函数会根据中间代码的类型（struct InterCode中的枚举类型kind）再调用某个具体的assembleX指令，这里重点是如何理解一些栈中内存的分配（想象push、pop和param、call等操作该如何实现，可以结合“计算机系统基础”中的汇编知识），以及如何针对不同的操作数类型（常数、变量和求值、取地址操作）写指令。

1. 反思与总结
   1. 自己编写的测试样例覆盖面还是非常有限，这次又注意到实验3的一个bug，即在函数translateCond中只考虑Exp RELOP Exp，NOT Exp，Exp AND/OR Exp等情况而没有列入LP Exp RP，类似这样的错误，整个实验中不知道还有多少。
2. 编译运行

使用makefile进行编译。直接cd到Code目录下make即可。

编译完成后，输入./parser test output对test进行分析，汇编代码输出到output中。