实验 4　　191220154　　张涵之

a) 实现功能
   a) 必做内容：实现目标代码的生成；

b) 实现思路
   a) 在实验 3 基础上继续扩展 lib.h 和 lib.c 中定义的数据结构：

```
typedef enum {
    LI_, MOVE_, ADDI_, ADD_, SUB_, MUL_, DIV__,
    LW_, SW_, BEQ_, BNE_, BGT_, BLT_, BGE_, BLE_
} COMMAND_;

typedef struct Memory_* Memory;

struct Memory_ {
    char name[32];
    int fpOffset;
    Memory link;
};
```

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。

```
void initMemoryBlock();
Memory newMemory(char* name, int fpOffset);
void insertMemory(Memory mem);
Memory lookUpBlock(char* name);
char* findFreeTReg();
void freeAllTReg();
char* findLeftReg(FILE *p, Operand op);
char* findRightReg(FILE *p, Operand op);
void genAssemblyCode(FILE* p, COMMAND_ c, ...);
```

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。

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

```
void assembly(char* filename);
void assembleCode(FILE* p, InterCode code);
void assembleLabel(FILE* p, InterCode code);
void assembleFunc(FILE* p, InterCode code);
void assembleAssign(FILE* p, InterCode code);
void assembleBinary(FILE* p, InterCode code);
void assembleGoto(FILE* p, InterCode code);
void assembleIfgoto(FILE* p, InterCode code);
void assembleReturn(FILE* p, InterCode code);
void assembleDec(FILE* p, InterCode code);
void assembleArg(FILE* p, InterCode code);
void assembleCall(FILE* p, InterCode code);
void assembleParam(FILE* p, InterCode code);
void assembleRead(FILE* p, InterCode code);
void assembleWrite(FILE* p, InterCode code);
```

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。

c) 反思与总结
   a) 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。

d) 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。