《编译技术》课程设计文档

学号：_____15061063_____

姓名：_____胡梓义_____

2018 年　　1 月　　13 日

一．需求说明

1．文法说明

1. ＜加法运算符＞ ::= +|-

范例：+

分析：定义了加法运算符，独立的’+’，’-’，在出现加法运算符定义时，一次只能选择一个加法运算符且不能为空。

2. ＜乘法运算符＞ ::= *|/

范例：*

分析：定义了乘法运算符，独立的’*’，’/’，在出现乘法运算符定义时，一次只能选择一个乘法运算符且不能为空。

3. ＜关系运算符＞ ::= <|<=|>|>=|!=|==

范例：<

分析：定义了关系运算符，分别是独立
的’<’,’<=’,’>’,’>=’,’!=’.’==’，在出现关系运算符定义时，一次只能选择一个关系运算符且不能为空。

4. ＜字母＞ ::= __|a|...|z|A|...|Z

范例：a

分析：定义了字母，由26个小写字母26个大写字母和下划线独立组成，下划线定义函数名的时候可能会使用，比如_foo（）函数，在出现字母定义时，一次只能选择一个字母或下划线并且不能为空。

5. ＜数字＞ ::= 0 |＜非零数字＞

范例：1

分析：定义了数字，数字由0或非零字母组成，因为整数等一般都为非前0数字，所以将数字与非零数字分开定义，在出现数字定义时，一次只能选择一个数字并且不能为空。

6. ＜非零数字＞ ::= 1 |...| 9

范例：定义了非零数字，有1-9组成，在出现非零数字定义时，一次只能选择一个数字且不能为空。

7. ＜字符＞ ::= ‘＜加法运算符＞’|’＜乘法运算符＞’|’＜字母＞’|’＜数字＞’

范例：’ +’

分析：定义了字符，字符由加法运算符、乘法运算符、字母或数字单独加单引号组成，在出现字符定义时，一次只能选择一种情况且不能为空。

8. ＜字符串＞ ::= “｛十进制编码为32,33,35-126的ASCII字符｝″

范例：“abc”

分析：定义了字符串，字符串由若干十进制编码为32,33,35-126的ASCII字符加双引号组成，{}表示{}号中的元素可以出现0到无穷次，所以字符串可以为空串比如“”。

9. ＜程序＞ ::= ［＜常量说明＞］［＜变量说明＞］{＜有返回值函数定义＞|＜无返回值函数定义＞}＜主函数＞

范例：const int a = 1, b = 2;

```
int q, s[2];
int with(){
    return 5;
}
void without(int x, int y){
    x = y * 4;
}
void main(){
    q = with();
    without(a, b);
}
```

分析：定义了程序，程序中一定要有的是主函数，[]表示[]中的内容可以出现0次或1次，所以常量说明和变量说明可以没有，若有只能出现一次，有返回值的函数定义和无返回值的函数也可以不出现，出现则可以出现任意次，且两种函数定义顺序没有

要求。但四种成分若出现则顺序不能改变，先是常量变量说明再是函数定义再是主函数。

10. ＜常量说明＞ ::= const＜常量定义＞;{ const＜常量定义＞;}

范例：const int a = 1;

const char b = ‘w’;

分析：定义了常量说明，常量说明至少由一组const加常量定义组成，可以由1到任意组组成。

11. ＜常量定义＞ ::= int＜标识符＞＝＜整数＞{,＜标识符＞＝＜整数＞} | char＜标识符＞＝＜字符＞{,＜标识符＞＝＜字符＞}

范例：int a = 1， b = 2

分析：定义了常量定义，常量定义由int型的常量或char型的常量定义组成，声明int或char的类型后，至少有一组＜标识符＞＝＜整数＞或＜标识符＞＝＜字符＞，其后还可以追加，用逗号隔开。常量定义时必须为标识符赋值。在出现常量定义时，一次只能选择一种类型的定义且不能为空。

12. ＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝

范例：135

分析：定义了无符号整数，无符号整数不能有前零，所以第一位数字应为非零数字，后面可以接任意个数字也可以不接，无符号整数不能为0。

13. ＜整数＞ ::= ［＋｜－］＜无符号整数＞｜ 0

范例：+12

分析：整数分为有符号整数和无符号整数，正负号可以不出现但若出现只能单独出现一次，后接无符号整数，0也属于整数，有符号0也属于整数。0和无符号整数一次不能同时出现。

14. ＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

范例：asd0

分析：定义了标识符，标识符至少由一个字母组成，后还可以跟任意个字母或数字,标识符首位必须为字母。

15. ＜声明头部＞ ::= int＜标识符＞ |char＜标识符＞

范例：int a

分析：定义了声明头部，由int或char加标识符组成，出现声明头部定义时，一次只能选择一种类型的声明且不能为空。

16. ＜变量说明＞　　::=　＜变量定义＞;{＜变量定义＞;}

范例：int a;

　　int b;

分析：定义了变量说明，变量说明至少由一组变量定义组成，后可跟任意组变量定义，由分号隔开。

17. ＜变量定义＞　::=　＜类型标识符＞(＜标识符＞|＜标识符＞‘[’＜无符号整数＞‘]’){,(＜标识符＞|＜标识符＞‘[’＜无符号整数＞‘]’)}

范例：int a, s[3]

分析：定义了变量定义，变量定义类型标识符加标识符或标识符加’[’无符号整数’]’组成，在定义类型标识符后，至少有一组标识符或标识符加’[’无符号整数’]’，其后可用逗号隔开进行任意组的定义。标识符加’[’无符号整数’]’是进行了数组的定义。在进行变量定义时，不能为变量赋值。

18. ＜常量＞　　::=　＜整数＞|＜字符＞

范例：45

分析：定义了常量，常量由整数或字符组成，在出现常量的定义时，一次只能选择一种成分且不能为空。

19. ＜类型标识符＞　　::=　int | char

范例：int

分析：定义了类型标识符，类型标识符由int或char构成，在出现类型标识符定义时，一次只能选择一种类型且不能为空。

20. ＜有返回值函数定义＞　::=　＜声明头部＞‘(’＜参数＞‘)’　‘{’＜复合语句＞‘}’|＜声明头部＞‘{’＜复合语句＞‘}’　//第一种选择为有参数的情况，第二种选择为无参数的情况

范例：int a(){

　　int x, y;

　　y = x * 2

　}

分析：定义了有返回值函数定义，首先必须有声明头部，声明头部中的类型标识符即为该函数返回值的类型，该定义有两种选择，第一种是由参数的情况，第二种是无参数的情况，第一种情况声明头部紧接的小括号中有参数声明，第二种没有，其后再跟上中括号内加上复合语句。在出现有返回值函数定义的定义时，一次只能选择一种选择且不能为空。根据复合语句的定义，变量常量的说明必须在函数体内的开头。

21. ＜无返回值函数定义＞　::= void＜标识符＞(’＜参数＞‘)’ ‘{’＜复合语句＞‘}’| void＜标识符＞{’＜复合语句＞’}’//第一种选择为有参数的情况，第二种选择为无参数的情况

范例：void b(int x, int y){

```
    y = x;
}
```

分析：定义了无返回值函数定义，首先是void加标识符定义“声明头部”，该定义有两种选择，第一种是由参数的情况，第二种是无参数的情况，第一种情况“声明头部”紧接的小括号中有参数声明，第二种没有，其后再跟上中括号内加上复合语句。在出现无返回值函数定义的定义时，一次只能选择一种选择且不能为空。根据复合语句的定义，变量常量的说明必须在函数体内的开头。

22. ＜复合语句＞　::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞

范例：const x = 3;

```
int y,  z;
y = x * 3;
z = y *x;
```

分析：定义了复合语句，复合语句由常量说明，变量说明和语句列组成，其中常量说明和变量说明可以出现0到1次，即可以不进行常量说明和变量说明，若进行，只能出现一次，且先常量说明后变量说明再语句列的顺序不能改变。

23. ＜参数＞　::= ＜参数表＞

范例：int x, int y

分析：定义了参数，参数由参数表构成。

24. ＜参数表＞　::= ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}

范例：int x, int y

分析：定义了参数表，参数表至少由一组类型标识符加标识符组成，后可跟任意组类型标识符加标识符，用逗号隔开。

25. ＜主函数＞　　::=　void main’(’ ‘)’ ‘{’＜复合语句＞‘}’

范例：void main(){

```
    const x = 3;
    int y,  z;
    y = x * 3;
    z = y *x;
}
```

分析：定义了主函数，主函数可以看作一个特殊的无返回值函数定义，只是其标识符必须为main且无参数。根据复合语句的定义，变量常量的说明必须在函数体内的开头。

26. ＜表达式＞　　::=　［＋｜－］＜项＞{＜加法运算符＞＜项＞}

范例：+a

分析：定义了表达式，首先出现的正负号可以出现0或1次，即可以不出现但出现只能出现一次，后接项，表达式中至少有一个项，项后还可接任意组加法运算符加项，加法运算符作为项与项之间的连接，项与项之间只能由加法运算符连接。

27. ＜项＞　　::=　＜因子＞{＜乘法运算符＞＜因子＞}

范例：a * b

分析：定义了项，项至少由一个因子构成，后可跟任意组乘法运算符加因子，因子与因子之间由乘法运算符连接，也只能由乘法运算符连接。

28.　＜因子＞　　::=　＜标识符＞｜＜标识符＞‘[’＜表达式＞‘]’｜‘(’＜表达式＞‘)’｜＜整数＞|＜字符＞｜＜有返回值函数调用语句＞

范例：s[a+b]

分析：定义了因子，由标识符或标识符’[’表达式’]’或’(’表达式‘)’或整数或字符或有返回值函数调用语句组成，出现因子定义时，一次只能选择一种选项且不能为空。

29. ＜语句＞　　::=　＜条件语句＞｜＜循环语句＞|　‘{’＜语句列＞‘}’｜＜有返回值函数调用语句
＞;　　|＜无返回值函数

调用语句>;|<赋值语句>;|<读语句>;|<写语句>;|<空>;|<情况语句>|<返回语句>;

范例：if(x>y) x = y;

else y = x;

分析：定义了语句，由条件语句或循环语句或{语句列}或有返回值函数调用语句;或无返回值函数调用语句;或赋值语句;或读语句;或写语句;或情况语句或返回语句;组成，也可为空但要接分号，出现语句定义时，一次只能选择一种。在赋值语句、读语句、写语句、返回语句、有返回值函数调用语句和无返回值函数调用语句这几种简单语句后要跟分号，其他类型的语句在定义中都包含了这些简单语句或者可以选择<空>;来加上分号。

30. <赋值语句>　　::=　<标识符>=<表达式>|<标识符>’[’<表达式>‘]’=<表达式>

范例：a = 2

分析：定义了赋值语句，由标识符 = 表达式或标识符’[’表达式’]’=表达式组成，后者为数组赋值，在出现赋值语句定义时，一次只能选择一种组成且不能为空。

31. <条件语句>::= if ‘(’<条件>’)’<语句>else<语句>

范例：if(x>y) x = y;

else y = x;

分析：定义了条件语句，条件语句一定是if-else的形式。

32. <条件>　　::=　<表达式><关系运算符><表达式>|<表达式>　//表达式为0条件为假，否则为真

范例：x > y

分析：定义了条件，由表达式加关系运算符加表达式或表达式两种形式组成，第二种情况当表达式为0时条件为假否则为真，在出现条件定义时，一次只能选择两种形式的一种且不能为空。

33. <循环语句>　　::=　while ‘(’<条件>’)’<语句>

范例：while(i < 100){

a+=b;

i++;

}

分析：定义了循环语句，由固定句式while(条件)后接语句组成。

34. ＜情况语句＞　::=　switch ‘(’＜表达式＞‘)’ ‘{’＜情况表＞[＜缺省＞] ‘}’

范例：switch(i){

case 1 : x = y;

case 2 : y = x;

default : i = 0;

}

分析：定义了情况语句，由switch(表达式){}的固定句式组成，大括号中为情况表，缺省情况可以不出现，但如出现只能在情况表后出现一次。

35. ＜情况表＞　::=　＜情况子语句＞{＜情况子语句＞}

范例：case 1 : x = y;

case 2 : y = x;

分析：定义了情况表，情况表至少由一组情况子语句组成，后还可以接任意组情况子语句。

36. ＜情况子语句＞　::=　case＜常量＞：＜语句＞

范例：case 2 : y = x;

分析：定义了情况子语句，情况子语句由固定句式case 常量 ：语句构成。

37. ＜缺省＞　::=　default : ＜语句＞

范例：default : i = 0;

分析：定义了缺省语句，由固定句式default : ＜语句＞构成。

38. ＜有返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’|<标识符>
//第一种选择为有参数的情况，第二种选择为无参数的情况

范例：s(a+b)

分析：定义了有返回值函数调用语句，由标识符(值参数表)或标识符两种形式组成，第一种选择为有参数的情况，第二种选择为无参数的情况，在出现有返回值函数调用语句的定义时，一次只能选择一种形式且不能为空。

39. ＜无返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’|<标识符>
//第一种选择为有参数的情况，第二种选择为无参数的情况

范例：func

分析：定义了无返回值函数调用语句，由标识符(值参数表)或标识符两种形式组成，第一种选择为有参数的情况，第二种选择为无参数的情况，在出现无返回值函数调用语句的定义时，一次只能选择一种形式且不能为空。

40. ＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}

范例：a, b, a+b

分析：定义了值参数表，值参数表由至少一组表达式组成，后可接任意组表达式，表达式之间用逗号隔开。

41. ＜语句列＞ ::= {＜语句＞}

范例：a = x + y;

b = s(a, x);

分析：定义了语句列，语句列由任意组语句组成，可以为空，也可以有无数组语句。

42. ＜读语句＞ ::= scanf ‘(’＜标识符＞{,＜标识符＞}’)’

范例：scanf(a)

分析：定义了读语句，由固定句式scanf()的形式组成，小括号内至少有一组标识符，后可接任意组标识符，标识符之间由逗号隔开。

43. ＜写语句＞ ::= printf ‘(’ ＜字符串＞,＜表达式＞ ‘)’| printf ‘(’＜字符串＞ ‘)’| printf ‘(’＜表达式＞’)’

范例：printf(abc)

分析：定义了写语句，有printf(字符串,表达式)、printf (字符串)、printf (表达式)三种形式，在出现写语句定义时，一次只能选择一种形式且不能为空。

44. ＜返回语句＞ ::= return[‘(’＜表达式＞ ‘)’]

范例：return (x+y)

分析：定义了返回语句，开头有固定关键字return，后可以不接任何东西，若出现(表达式)只能出现一次。

附加说明：

（1）char类型的表达式，用字符的ASCII码对应的整数参加运算，在写语句中输出字符

（2）标识符不区分大小写字母

（3）写语句中的字符串原样输出

（4）情况语句中，switch后面的表达式和case后面的常量只允许出现int和char类型；每个情况子语句执行完毕后，不继续执行后面的情况子语句

（5）数组的下标从0开始

2. 目标代码说明

目标代码为 MIPS 指令代码，具体细节详见第二部分。

3. 优化方案*

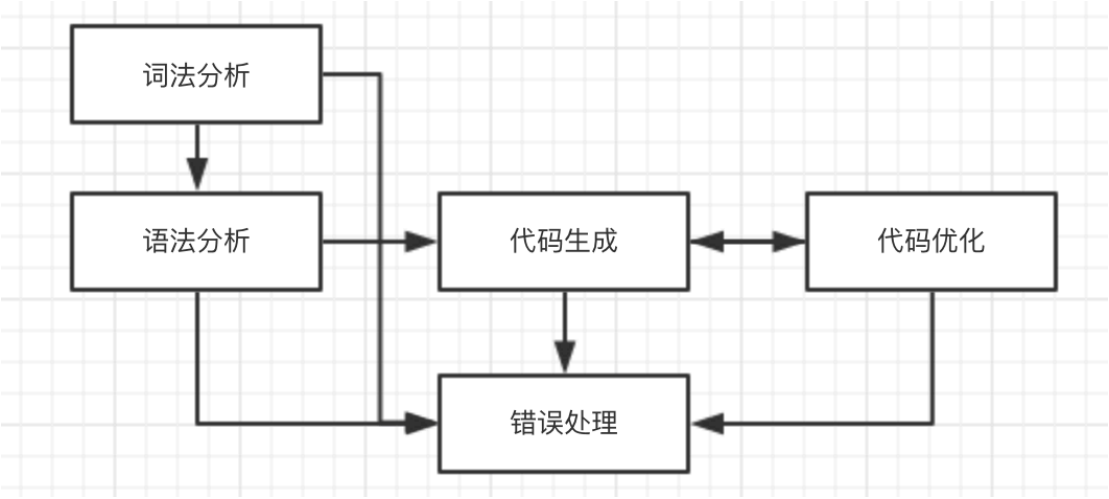
基本块内部的公共子表达式删除（DAG图）

全局寄存器分配（着色算法）

数据流分析（通过活跃变量分析建立冲突图）

二. 详细设计

1. 程序结构



2. 类/方法/函数功能

main.c：

函数	功能
int main()	入口函数，初始化

insymbol.c 词法分析：

函数	功能
int isLetter(char c)	判断当前字符是否是字母
int isDigit(char c)	判断当前字符是否是数字
char toLower(char c)	将大写字母转化为小写字母
int isStr(char c)	判断当前字符是否符合字符串编码要求
int toDigit(char str[1000])	将字符串转化为数字
void insymbol()	词法分析处理程序，分析符合文法要求的程序，并输出单词的类别码和单词值

lrparser.c 语法分析：

函数	功能
enum SYMBOL constDec(int lev)	分析常量申明
void program()	分析程序
void constDef(enum SYMBOL per, int lev)	分析常量定义
void varDef(enum SYMBOL temp, int lev)	分析变量定义
void defHead()	分析声明头部
void paramList()	分析参数表
void compoundstatement()	分析复合语句
void statementList()	分析语句列
void statement()	分析语句
void expression()	分析表达式
void factor()	分析因子
void term()	分析项
void ifstatement()	分析 if 语句
void require()	分析条件语句
void whilestatement()	分析 while 语句
void switchstatement()	分析 switch 语句
struct table casestatement(struct table label1, struct table label2, struct table label3, struct table place2, struct table place3)	分析 case 语句
void defaultstatement(struct table label1)	分析 default 语句
int vParamList(int i)	分析值参数表
void printfstatement()	分析写语句
void scanfstatement()	分析读语句
void returnstatement()	分析返回语句

table.c 符号表：

函数	功能
int enter(char* name, int kind, int type, int adr, int normal, int size, int lev)	登录符号表
int locSym(char* id, int fn)	查找 id 在符号表中的位置

midcode.c 语义分析生成四元式：

函数	功能
struct table newreg(int kind, int type, int adr)	增加新的临时变量
struct table newtag()	增加新的 label
void emit(char op[], struct table a, struct table b, struct table res)	生成四元式
struct table equal(struct table a, struct table b)	将 table b 赋给 table a
int ifequal(struct table a, struct table b)	判断两个 tabel 结构是否相等
struct table init(struct table a)	table 结构的初始化
int clear()	每个函数结束后确定该函数中符号（包括临时）所占空间及索引
void printcode()	打印四元式

mips.c 目标代码生成：

函数	功能
void getMips()	生成目标代码入口函数，处理一开始栈的分配及全局申明，并根据四元式判断对应的生成函数
void data()	生成. data 段代码
void genJump(int ins, int rs, char *label)	生成跳转代码
void store(int a, int st, int base)	生成 store 代码
void load(int a, int ld, int base)	生成 load 代码
void loadgp(int a, char gp[1000])	生成全局变量或常量的 load 代码
void storegp(int a, char gp[1000])	生成全局变量或常量的 store 代码
void genconst(struct middlecode mcode)	生成常量声明部分代码
void add_sub(int op, int a, int b, int res);	生成 addu/subu 运算代码
void add_sub_i(int op, int a, int imm, int res)	生成 addi/subi 运算代码
void mul_div(int op, int a, int b, int res)	生成 mul/div 运算代码
void sll_srl(int op, int a, int b, int imm)	生成 sll/srl 运算代码
void moveto(int res, int a)	生成 move 运算代码
void compare(struct middlecode code, int a, int b, int f)	分析比较类型四元式，生成分支语句代码
void condition_jump(int op, int a, int b, char lab[1000])	根据分支语句情况生成条件跳转语句代码
void calculate(struct middlecode mcode, int f)	分析运算类型四元式，生成运算语句代码
void call(struct middlecode mcode, int f)	分析函数调用类型四元式，生成函数调用

	相关代码
void genreturn(struct middlecode mcode, int f)	分析返回语句类型四元式，生成返回语句相关代码
void genscanf(struct middlecode mcode, int f)	分析读语句类型四元式，生成读语句相关代码
void genprintf(struct middlecode mcode, int f)	分析写语句类型四元式，生成写语句相关代码

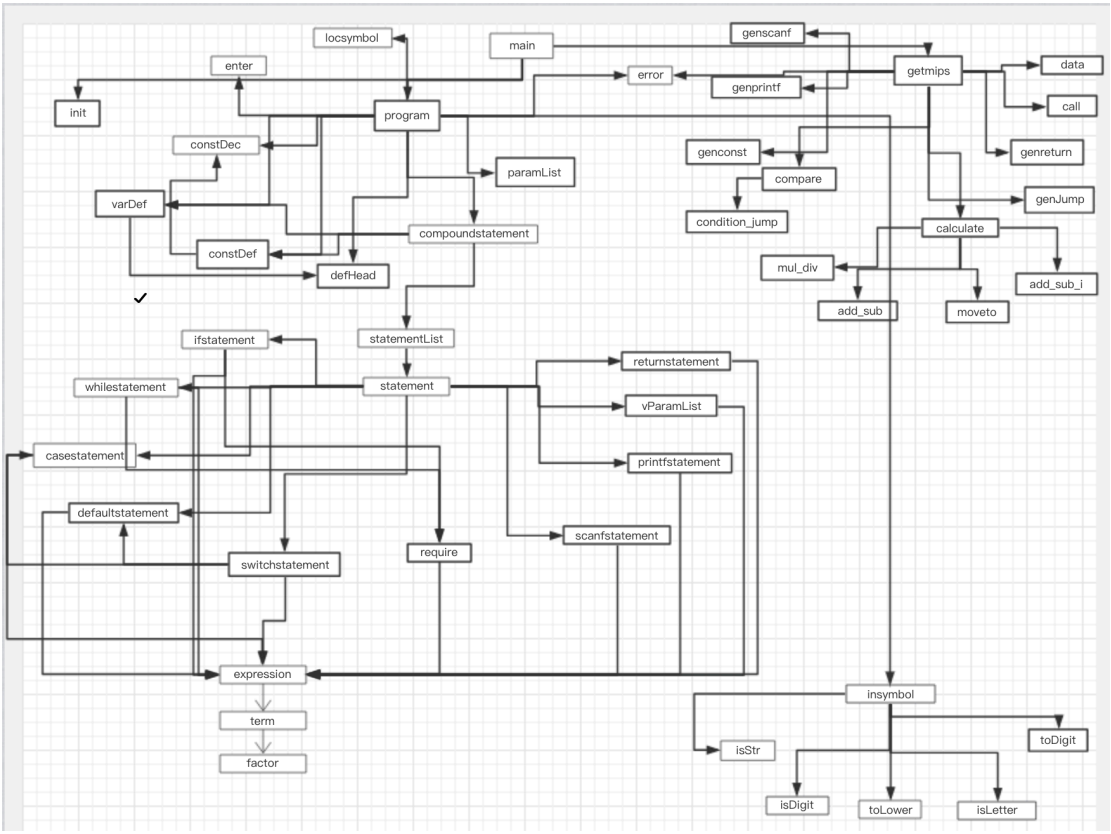
midcode.c 代码优化：

函数	功能
void dag()	进行 dag 图优化
void activity()	通过活跃变量分析建立冲突图
void color()	根据冲突图进行着色算法

error.c 错误处理：

函数	功能
void error(int n)	错误处理，显示出错行数并打印错误信息

3. 调用依赖关系



4. 符号表管理方案

标识符表

```
struct table{           //标识符表
    char name[1000];
    int kind;           //0常量，1变量，2函数，3array
    int type;           //0int，1char
    int ref;            //对于数组，记录其首地址；对于函数，记录其入口地址；其余为0；对于类型，记录其所需单元数目；对于常量填入值；对于形参or变量，填入其在运行栈中的位置。
    int adr;            //对于数组，记录其在atab中的位置；对于函数，记录其在ftab中的位置。
    int normal;         //标识符是否是函数形参
    int size;
    int lev;
    char arr[1000];
    int func;
    int reg;
};
```

数组属性表

```
struct arr{                 //数组表
    int high;
    int size;
    int type;
    char arrname[100];
    char ref[1000];
};
```

函数登录表

```
struct ifRetFunc{               //函数表
    int flag;               //0表示无返回值函数
    char funcname[100];
    int retFlag;                //1表示函数中有return
    int size;
    int lastpar;
    int last;
    int parnum;
    int type;
    int regnum;
};
```

字符串登录表

```
char stab[1000][1000];                    //字符串表
```

5. 存储分配方案

优化前：

栈式动态存储分配，从首地址开始依次分配常量变量等，程序有一个运行栈，每次函数调用有一个独立的运行栈，遇函数返回则退出运行栈。

优化后：

根据全局寄存器的分配，将某些分配到寄存器的变量存在寄存器中，其余值存在运行栈中。

运行栈结构：

局部数据区　　//存储局部变量
显示参数区　　//函数调用参数
隐式参数区　　//返回地址 ret addr、指向前一个活动记录基的指针 prev abp、返回值 ret value
Display 区

6. 解释执行程序*

本程序不采用解释执行程序，而是编译执行。

7. 四元式设计*

```
struct middlecode{
    char op[10];
    struct table a;
    struct table b;
    struct table res;
};
```

	op	a	b	res
ADDU/ADDI	+	a	b	res
SUBU/SUBI	-	a	b	res

MUL	*	a	b	res
DIV	/	a	b	res
ASSIGN	=	a		res
CALL	call	a		
JR	jr			addr
BEQ	==	a	b	label
BNE	!=	a	b	label
BGEZ	>=	a	b	label
BGTZ	>	a	b	label
BLEZ	<=	a	b	label
BLTZ	<	a	b	label
SETLABEL	setlabel			label
JAL	jal	a		label
RETURN	return			res
PRINTF	printf	a	b	res
SCANF	scanf	a	b	res
CONST	const	a	b	res
INT	int	a	b	res
START	start			funcname
END	end			funcname

8. 目标代码生成方案*

四元式 -→ mips 指令结构

在 mips.c 中的 getmips 函数中，首先调用 data 函数进行全局声明的执行，再根据顺序循环每个函数，在每个函数开始的地方进行运行栈的空间分配，在函数内部根据每一条四元式的 op 值判断应该进入到哪个函数进行代码生成，如：若四元式为+,a,b,r，则进入 calculate 函数进行运算。在负责翻译的函数内部通过进入到函数的四元式的结构进行目标代码翻译。

9. 优化方案*

基本块内部的公共子表达式删除（DAG图）

全局寄存器分配（着色算法）

数据流分析（通过活跃变量分析建立冲突图）

10. 出错处理

分析遇错误时调用 error 函数，显示出错行数并输出错误信息。

```
void error(int n){
    char *msg[] = {
        "wrong identify",        //0非法标识符
        "wrong char",        //1非法字符
        "wrong string",      //2非法字符串
        "single quote",      //3单引号缺失
        "double quote",      //4双引号缺失
        "unknown",       //5非法符号
        "right parent",      //6小括号缺失
        "left parent",
        "right brace",       //8大括号缺失
        "left brace",
        "right bracket",         //10中括号缺失
        "left bracket",
        "!= without a =",        //12
        "expect a number",       //13
        "function without a return value",       //14
        "function with a return value",      //15
        "expected semicolon",          //16
        "constant type",             //17
        "expect a constant",              //18
        "right side of equal sign is wrong",          //19
        "expected initialization of const",               //20
        "wrong key word",             //21
        "expect a function with return value",            //22
        "expect a else",              //23
        "expect a colon",                 //24
        "expect an expression",           //25
        "identify redefine",              //26标识符已存在
        "parameter number doesn't match",             //27参数不匹配
        "identify undefine",                //28标识符未定义
        "constant can not be assigned",           //29
        "function can not be assigned"              //30
    };
    printf("error at line %d: %s\n", line, msg[n]);
}
```

至上而下分别为 error(0)到 error(30).

三．操作说明

1．运行环境

运行环境为 64 位 Win10 系统，运行软件为 CodeBlocks13.12 版本。

2．操作步骤

运行代码后在编译器控制台输入目标编译文件的完整路径回车运行，优化前的目标代码结果保存在工程目录下的 result.txt 文件中，优化后的目标代码保存在工程目录下的 result2.txt 文件中

四．测试报告

1．测试程序及测试结果

test1.txt:

输入：01

100

输出：

%d 1%d 2%d 3%d 4%d 5
%d 1%d 2%d 3%d 5%d 4
%d 1%d 2%d 4%d 3%d 5
%d 1%d 2%d 4%d 5%d 3
%d 1%d 2%d 5%d 4%d 3
%d 1%d 2%d 5%d 3%d 4
%d 1%d 3%d 2%d 4%d 5
%d 1%d 3%d 2%d 5%d 4
%d 1%d 3%d 4%d 2%d 5
%d 1%d 3%d 4%d 5%d 2
%d 1%d 3%d 5%d 4%d 2
%d 1%d 3%d 5%d 2%d 4
%d 1%d 4%d 3%d 2%d 5
%d 1%d 4%d 3%d 5%d 2
%d 1%d 4%d 2%d 3%d 5
%d 1%d 4%d 2%d 5%d 3
%d 1%d 4%d 5%d 2%d 3
%d 1%d 4%d 5%d 3%d 2
%d 1%d 5%d 3%d 4%d 2
%d 1%d 5%d 3%d 2%d 4
%d 1%d 5%d 4%d 3%d 2
%d 1%d 5%d 4%d 2%d 3
%d 1%d 5%d 2%d 4%d 3
%d 1%d 5%d 2%d 3%d 4
%d 2%d 1%d 3%d 4%d 5
%d 2%d 1%d 3%d 5%d 4
%d 2%d 1%d 4%d 3%d 5
%d 2%d 1%d 4%d 5%d 3
%d 2%d 1%d 5%d 4%d 3
%d 2%d 1%d 5%d 3%d 4
%d 2%d 3%d 1%d 4%d 5
%d 2%d 3%d 1%d 5%d 4
%d 2%d 3%d 4%d 1%d 5
%d 2%d 3%d 4%d 5%d 1
%d 2%d 3%d 5%d 4%d 1
%d 2%d 3%d 5%d 1%d 4
%d 2%d 4%d 3%d 1%d 5
%d 2%d 4%d 3%d 5%d 1
%d 2%d 4%d 1%d 3%d 5
%d 2%d 4%d 1%d 5%d 3

%d 2%d 4%d 5%d 1%d 3
%d 2%d 4%d 5%d 3%d 1
%d 2%d 5%d 3%d 4%d 1
%d 2%d 5%d 3%d 1%d 4
%d 2%d 5%d 4%d 3%d 1
%d 2%d 5%d 4%d 1%d 3
%d 2%d 5%d 1%d 4%d 3
%d 2%d 5%d 1%d 3%d 4
%d 3%d 2%d 1%d 4%d 5
%d 3%d 2%d 1%d 5%d 4
%d 3%d 2%d 4%d 1%d 5
%d 3%d 2%d 4%d 5%d 1
%d 3%d 2%d 5%d 4%d 1
%d 3%d 2%d 5%d 1%d 4
%d 3%d 1%d 2%d 4%d 5
%d 3%d 1%d 2%d 5%d 4
%d 3%d 1%d 4%d 2%d 5
%d 3%d 1%d 4%d 5%d 2
%d 3%d 1%d 5%d 4%d 2
%d 3%d 1%d 5%d 2%d 4
%d 3%d 4%d 1%d 2%d 5
%d 3%d 4%d 1%d 5%d 2
%d 3%d 4%d 2%d 1%d 5
%d 3%d 4%d 2%d 5%d 1
%d 3%d 4%d 5%d 2%d 1
%d 3%d 4%d 5%d 1%d 2
%d 3%d 5%d 1%d 4%d 2
%d 3%d 5%d 1%d 2%d 4
%d 3%d 5%d 4%d 1%d 2
%d 3%d 5%d 4%d 2%d 1
%d 3%d 5%d 2%d 4%d 1
%d 3%d 5%d 2%d 1%d 4
%d 4%d 2%d 3%d 1%d 5
%d 4%d 2%d 3%d 5%d 1
%d 4%d 2%d 1%d 3%d 5
%d 4%d 2%d 1%d 5%d 3
%d 4%d 2%d 5%d 1%d 3
%d 4%d 2%d 5%d 3%d 1
%d 4%d 3%d 2%d 1%d 5
%d 4%d 3%d 2%d 5%d 1
%d 4%d 3%d 1%d 2%d 5
%d 4%d 3%d 1%d 5%d 2
%d 4%d 3%d 5%d 1%d 2
%d 4%d 3%d 5%d 2%d 1

%d 4%d 1%d 3%d 2%d 5
%d 4%d 1%d 3%d 5%d 2
%d 4%d 1%d 2%d 3%d 5
%d 4%d 1%d 2%d 5%d 3
%d 4%d 1%d 5%d 2%d 3
%d 4%d 1%d 5%d 3%d 2
%d 4%d 5%d 3%d 1%d 2
%d 4%d 5%d 3%d 2%d 1
%d 4%d 5%d 1%d 3%d 2
%d 4%d 5%d 1%d 2%d 3
%d 4%d 5%d 2%d 1%d 3
%d 4%d 5%d 2%d 3%d 1
%d 5%d 2%d 3%d 4%d 1
%d 5%d 2%d 3%d 1%d 4
%d 5%d 2%d 4%d 3%d 1
%d 5%d 2%d 4%d 1%d 3
%d 5%d 2%d 1%d 4%d 3
%d 5%d 2%d 1%d 3%d 4
%d 5%d 3%d 2%d 4%d 1
%d 5%d 3%d 2%d 1%d 4
%d 5%d 3%d 4%d 2%d 1
%d 5%d 3%d 4%d 1%d 2
%d 5%d 3%d 1%d 4%d 2
%d 5%d 3%d 1%d 2%d 4
%d 5%d 4%d 3%d 2%d 1
%d 5%d 4%d 3%d 1%d 2
%d 5%d 4%d 2%d 3%d 1
%d 5%d 4%d 2%d 1%d 3
%d 5%d 4%d 1%d 2%d 3
%d 5%d 4%d 1%d 3%d 2
%d 5%d 1%d 3%d 4%d 2
%d 5%d 1%d 3%d 2%d 4
%d 5%d 1%d 4%d 3%d 2
%d 5%d 1%d 4%d 2%d 3
%d 5%d 1%d 2%d 4%d 3
%d 5%d 1%d 2%d 3%d 4
1

test2.txt:

输入：1
a

输出：111111asdfgha

test3.txt:

输入：9
w

输出：13w

test4.txt:

输入：4

w

输出：456717

test5.txt:

输入：3

输出：720

test1_error.txt:

输出：

error at line 1: right side of equal sign is wrong
error at line 11: identify undefine
error at line 13: identify undefine
error at line 20: identify undefine
error at line 26: identify undefine
error at line 26: parameter number doesn't match
error at line 82: parameter number doesn't match
error at line 83: right brace
error at line 84: expect a else
error at line 88: parameter number doesn't match
error at line 94: right brace

test2_error.txt:

输出：

error at line 1: left parent
error at line 2: left brace
error at line 3: left brace

test3_error.txt:

输出：

error at line 7: constant can not be assigned
error at line 9: identify undefine
error at line 15: expect a else

test4_error.txt:

输出：

error at line 5: wrong key word
error at line 13: expected semicolon
error at line 13: unknown
error at line 13: unknown
error at line 17: identify undefine
error at line 21: identify undefine
error at line 23: identify undefine
error at line 26: identify undefine
error at line 27: identify undefine

test5_error.txt:

输出：

error at line 13: expected semicolon
error at line 13: unknown
error at line 17: identify undefine
error at line 21: identify undefine
error at line 23: identify undefine
error at line 26: identify undefine

2．测试结果分析

1.test1(正确)：

本程序测试了函数调用，递归，常量变量声明，表达式，printf、scanf、return、if-else、while、switch-case 相关语句，赋值语句等。

程序的功能是先输入两个数，第一个为字符，第二个为数字，再根据输入选择是否继续输入并输出相应结果。当第一个输入字符为+、-、*、/时，会再输入两个数，进行相关计算并输出结果，当第一个字符为 0 时，继续输入一个数字，若其值大于字符 c 的 ascii 码值，输出 5 的全排列，否则输出 2 的全排列。最后的输出是输入的第二个数字。

2.test2(正确)：

本程序测试了常量变量申明，scanf、printf、switch-case 及其相关语句等。

程序的功能是，输入两个数，第一个为数字，第二个为字符，然后根据 switch 语句进行相关语句的输出。

3.test3(正确)：

本程序测试了常量变量申明，scanf、printf、if-else 及其相关语句等。

程序的功能是，输入两个数，第一个为数字，第二个为字符，然后根据 if-else 语句进行相关语句的输出。

4.test4(正确)：

本程序测试了常量变量申明，scanf、printf、if-else、while、return 及其相关语句，函数调用等。

程序的功能是，输入两个数，第一个为数字，第二个为字符，然后根据相关语句进行相关的输出。

5.test5(正确)：

本程序测试了常量变量申明，scanf、printf、if-else、while、return 及其相关语句，函数调用，递归等。

程序的功能是，输入一个数字，然后输出进行相关计算后的数字的阶乘结果。

6.test1_error(错误)：

本程序测试了给常量赋值类型不符，函数声明缺少应有参数，函数调用值参数数量不符等错误。

7.test2_error(错误)：

本程序测试了 main 函数后没有小括号等错误。

8.test3_error(错误)：

本程序测试了常量不能在声明之外的地方赋值，使用了未定义标识符，if 语句后没有 else 等错误。

9.test4_error(错误)：

本程序测试了 else 后缺少大括号或分号，return 后缺少小括号等错误。

10.test5_error(错误)：

本程序测试了 return 后缺少小括号，条件语句条件错误等错误。

五．总结感想

课程最开始的时候选择难度时还有犹豫，在经过计组和 oo 的折磨，对自己以及自己的承受能力不太自信。但最终还是选择了难度三。开始逐渐进入课设阶段后，刚开始的阶段任务不太重，让自己对这门课有了信心。同时也可以有一些时间去了解课设的内容，在期间也与同学分享交换了对于设计的一些想法，这种交流使我收获颇丰。词法分析和语法分析部分通过查阅相关资料自己都可以不错的完成，但在进入符号表设计的时候遇到了点困难，通过与同学的交流以及自己的思考，还是构造出了比较完备的符号表系统，通过写符号表我认识到了提前设计构思的重要性，符号表不能一拿到就想当然的开始写，而是需要对之前词法分析和语法分析部分进行了解以及对四元式生成和目标代码生成部分的需求进行分析之后再设计。否则写到后面可能就会由于符号表的不合理设计而走很多弯路甚至是需要重构代码，事实证明符号表的合理设计可以是后面的某些工作变得简单很多。在进行目标代码生成的设计时，一开始无从下手，在经过自己的学习和与同学的讨论后，我对 mips 运行的结构有了一个更深刻更正确的了解，这种了解是当初在计组学习时都没有达到的。到了后期事情开始增多的时候，课设还是带来了不小的压力，不过好在都正确的解决了，通过课设的学习也让我的 debug 能力和抗压能力有了显著提升。最值得一提的就是让我对构建并完善完成一个完整的工程项目有很大帮助，让我在以后的学习甚至工作生活中受益匪浅。