## Complier文档

开发小组：从容应队

1. **词法分析**
2. 使用工具

flex

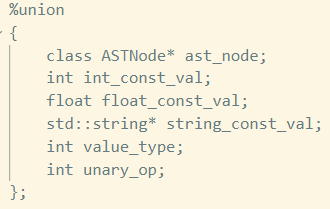
2、源文件

lexer.l

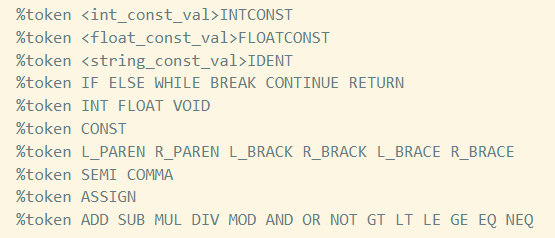
lexer.h

3、数据结构

（1）向语法分析器传递的数据



1. Token列表



4、功能说明

通过正则表达式描述词法单元，由flex工具生成词法分析器，返回tokens供语法分析器使用。

1. **语法分析**
2. 使用工具

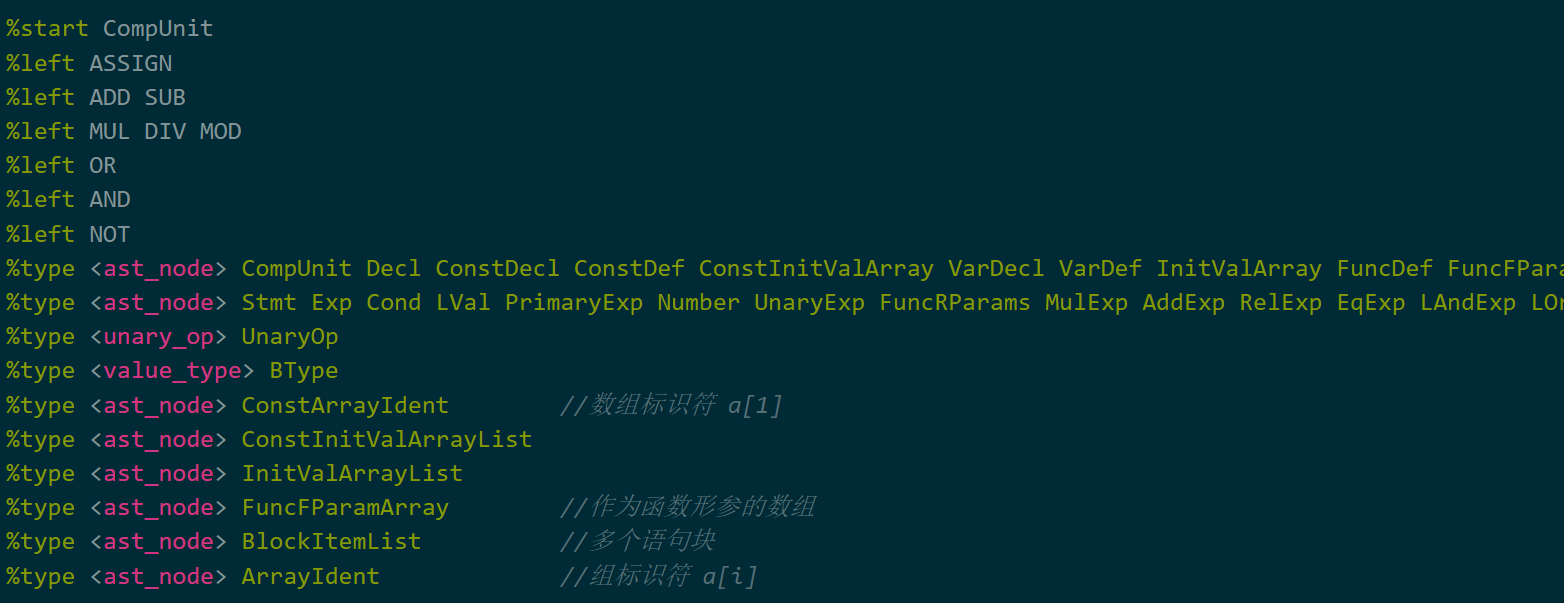
bison

1. 源文件

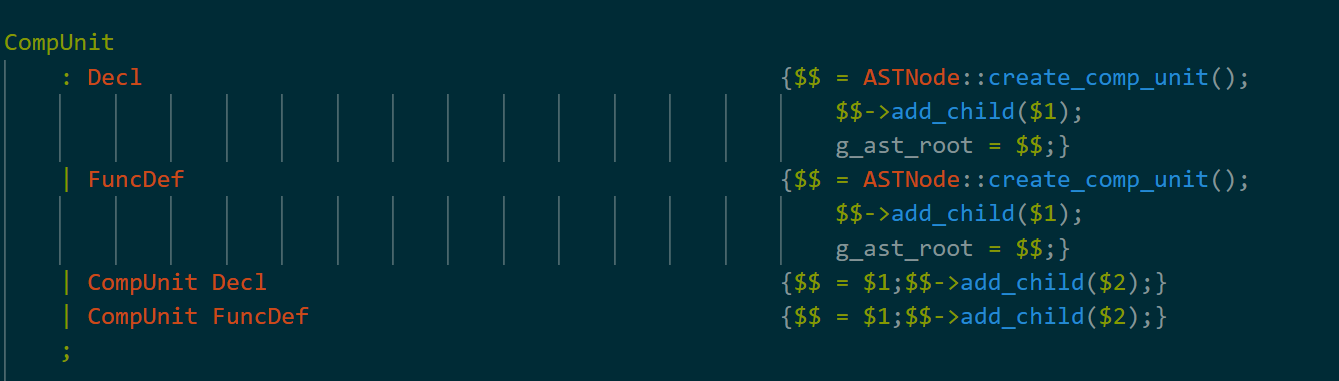
parser.y

1. 数据结构

（1）起点声明、结合性声明和类型声明



（2）推导文法（以CompUnit为例）



1. 功能说明

通过BNF表示描述文法，由bison工具生成语法分析器。

1. **语法制导翻译**
2. 源文件

parser.y

ast.h

ast.cpp

1. 数据结构
2. ASTNode

*//对应源程序的行号*

**int** m\_line\_no{-1};

*//子节点*

    std::vector<ASTNode \*> m\_child{{}};

*//基本类型*

    ASTType m\_type{ASTType::Unknown};

*//值类型 int或float*

    ValueType m\_value\_type{ValueType(BasicType::None)};

*//是否为左值*

**bool** m\_is\_lvalue{false};

*//是否为字面值*

**bool** m\_is\_literal{false};

*//常数值*

    BasicValue m\_value;

*//具体功能类型,有一些节点没有功能类型就是0*

**int** m\_func{0};

*//变量名(如果是Decl节点)*

    std::string m\_var\_name{""};

*//对应符号在符号表中的项*

    Symbol \*m\_symbol{nullptr};

1. 功能说明

在进行语法分析时，使用语义动作来构建程序的第一个中间表示形式——抽象语法树。

1. **语义检查**

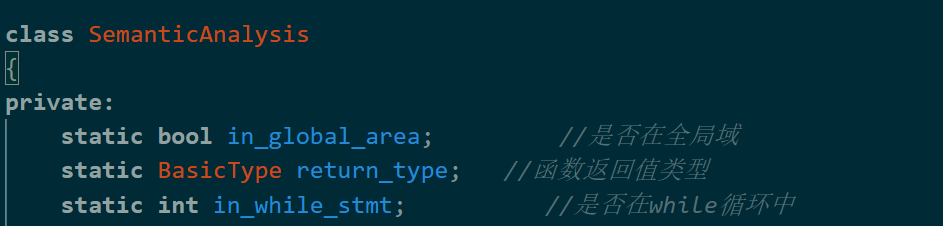
1、源文件

semantic\_analysis.h

semantic\_analysis.cpp

2、数据结构

（1）Semantic静态类



3、功能说明

（1）静态变量

in\_global\_area表示当前AST结点是否在全局域，从而对建立符号的类型进行设置。

return\_type表示一个函数返回值类型，在递归函数定义结点的子树时用于判断return语句的返回类型是否匹配。

in\_while\_stmt表示当前结点是否处于一个while语句结点的子树中，用于递归时检查break、continue语句的合法性。

（2）语义检查

功能：采用递归的方式从AST的根结点开始遍历，针对不同ASTType的结点进行相应的错误检查，根据需求选择先序遍历或后序遍历，同时建立符号表。

函数：semantic\_analysis

输入：AST

输出：SymbolTable（，报错）

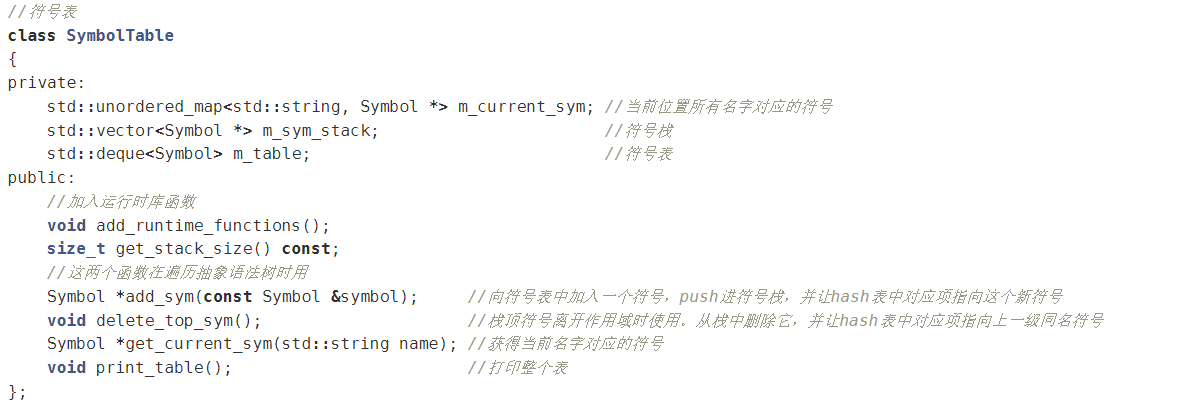
1. **符号表**
2. 源文件

symbol\_table.h

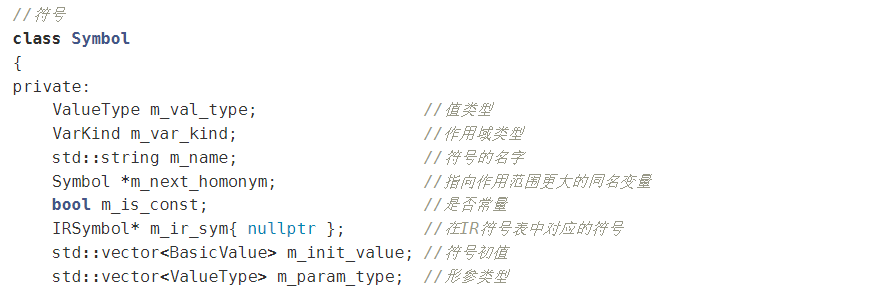
symbol\_table.cpp

1. 数据结构

（1）SymbolTable



（2）Symbol



1. 功能说明

符号表用于存放程序中出现的有关标识符的属性信息，由抽象语法树在语义分析中生成。

（1）SymbolTable

符号表类型中包括了符号栈、保存符号的双端队列以及存储当前程序位置对应符号的哈希表。

（2）Symbol

符号类中包括值类型、作用域类型、符号名称、常量标志、指向IR符号表中的指针、初值、形参等。

1. 添加符号

向符号表中加入一个符号，push进符号栈，并让哈希表中对应项指向该新符号

类：SymbolTable

函数：add\_sym

输入：Symbol类变量

输出：无

1. 删除符号

栈顶符号离开作用域时将其删除，并让哈希表指向上一级同名符号

类：SymbolTable

函数：delete\_top\_sym

输入：无

输出：无

1. 获取当前名字对应符号

类：SymbolTable

函数：get\_current\_sym

输入：字符串变量名

输出：Symbol类指针

1. **线性中间表示**
2. 源文件

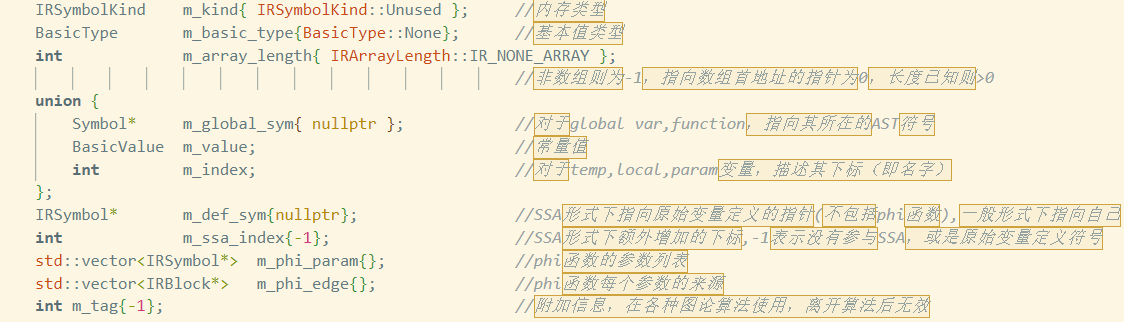
ir\_symbol\_table.h

ir\_symbol\_table.cpp

linear\_ir\_manager.h

linear\_ir\_manager.cpp

1. 数据结构
2. IRSymbol

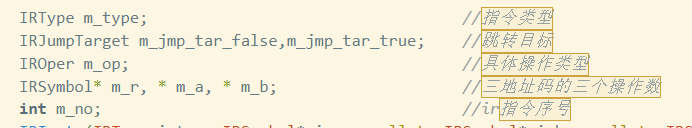


（2）IRSymbolTable



使用指针不易失效的deque维护IR符号表。

（3）IRInstr



（4）IRInstrList



1. 功能说明

（1）IRSymbol

IR符号，用于保存IR指令中的操作数。

（2）IRSymbolTable

IR符号表，用于分配、管理IR符号。

（3）IRInstr

表示一条IR指令，包含构造指令的工厂函数。

（4）IRInstrList

表示一串IR指令序列。

（5）IR的定义和约束

参见ir\_definition.md。

（6）线性IR生成

类：LinearIRManager

函数：gen\_ir

输入：AST

输出：线性IR

1. 线性IR打印

类：LinearIRManager

函数：print\_ir\_list

输入：线性IR、IO流

输出：将线性IR以字符流的形式打印到IO流上

1. **IR测试管理**
2. 源文件

ir\_test\_manager.h

ir\_test\_manager.cpp

1. 数据结构

无

1. 功能说明
2. IR转C语言

类：IRTestManager

函数：gen\_c\_source

输入：控制流图形式的IR

输出：可编译执行的C语言程序

目的：测试前端以及各种优化的功能正确性

1. **控制流图**
2. 源文件

cfg.h

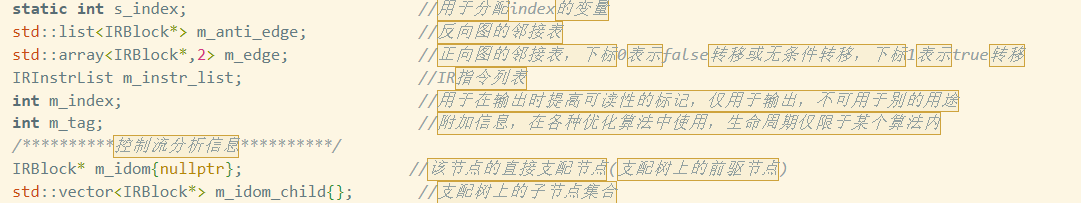
cfg.cpp

cfg\_managaer.h

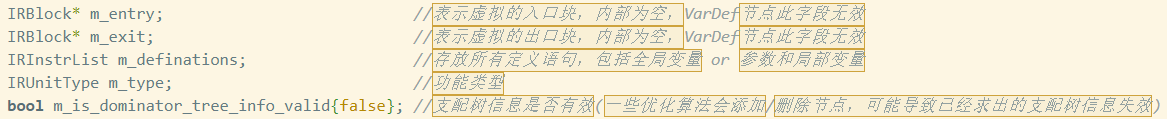
cfg\_manager.cpp

2、数据结构

（1）IRBlock



（2）IRUnit



（3）IRProgram



3、功能说明

（1）IRBlock

控制流图中的基本块，要么为空（entry和exit两个特殊基本块），要么尾部是一条跳转指令。

（2）IRUnit

描述一个编译单元，要么为函数，要么为全局变量定义列表

（3）IRProgram

描述由IR构成的整个程序

（4）控制流分析

包含针对IRUnit的支配树求解函数，构建基本块上的支配树。

类：CFGManager

函数：build\_dominator\_tree

输入：IRUnit

输出：支配树信息

1. 控制流图生成

类：CFGManager

函数：gen\_from\_ast/gen\_from\_linear\_ir

输入：AST/线性IR

输出：控制流图

1. 控制流图打印

模块：CFGManager-print\_ir\_program

输入：IRProgram、IO流

输出：以字符流的形式展示出数据流图

1. **寄存器分配**

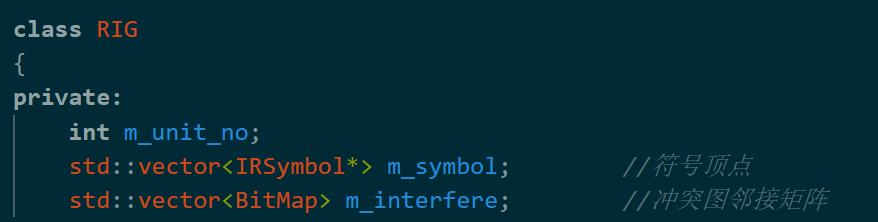
1、源文件

reg\_alloc.h

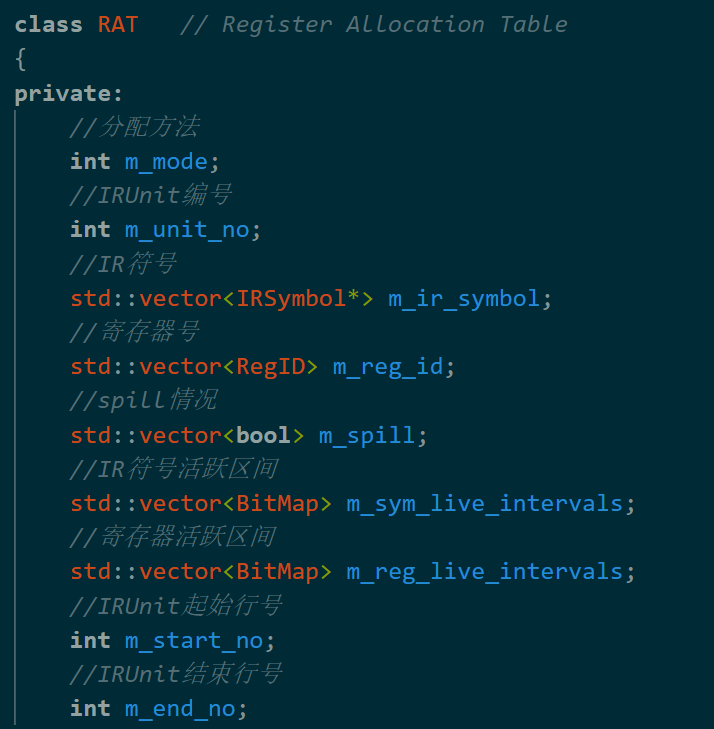
reg\_alloc.cpp

2、数据结构

(1)冲突图



(2)寄存器分配表



(3)进行寄存器分配的RegisterAllocator类



3、功能说明

寄存器分配实现了线性扫描和简单图着色两种方法。线性扫描方法对符号活跃区间进行标记并排序，然后按顺序扫描，维护一个当前活跃变量的记录，并分配寄存器号。简单图着色方法构建冲突图，然后循环进行化简和溢出，直到所有符号入栈，然后按序出栈进行冲突检测并分配寄存器号。

线性扫描相关函数：

(1)mark函数

功能：标记符号活跃信息的起始行号和结束行号，用于后续的排序和扫描

输入：符号活跃区间信息

输出：符号活跃区间信息

(2)sort函数

功能：按活跃区间起始行号大小对符号进行快速排序

输入：符号活跃区间信息

输出：符号活跃区间信息

(3)scan函数

功能：按线性方式逐行扫描，为符号指派寄存器或溢出。

输入：符号活跃区间信息

输出：寄存器分配表

图着色相关函数

(1)build\_graph函数

功能：按照活跃区间两两判定是否冲突，并构建冲突图

输入：符号活跃区间信息

输出：冲突图

(2)simplify函数

功能：对冲突图进行化简，将不影响图着色的结点压入select栈。

输入：冲突图

输出：冲突图

(3)spill

功能：对冲突中无法化简的结点进行溢出操作，将其压入spill栈。

输入：冲突图

输出：冲突图

(4)select

功能：根据冲突图、select栈、spill栈中的信息分配寄存器。

输入：冲突图

输出：寄存器分配表

1. **活跃变量分析**

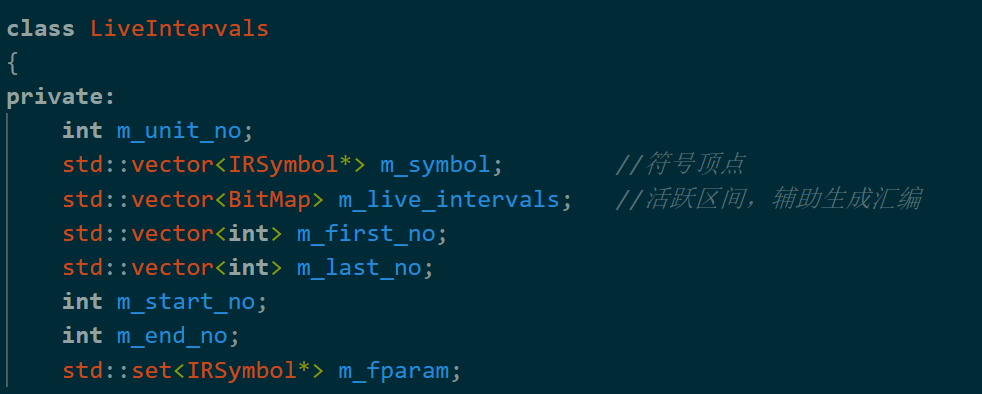
1、源文件

reg\_alloc.h

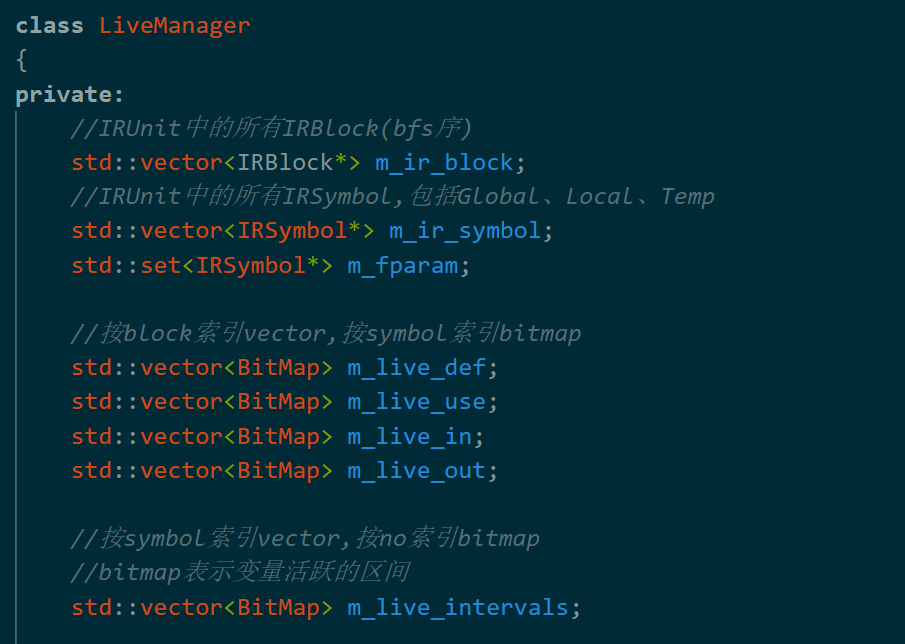
reg\_alloc.cpp

2、数据结构

(1)符号活跃区间信息



(2)进行活跃变量分析的LiveManager类



3、功能说明

活跃变量分析的内容主要有：

①对IR指令按一定顺序进行标号，以此作用活跃区间表示的基准；

②收集IR单元中需要进行活跃变量分析的符号；

③计算每个基本块中定值和使用的变量；

④计算每个基本块的入口活跃和出口活跃变量；

⑤计算得到每个符号的活跃区间。

1. **静态单赋值形式**
2. 源文件

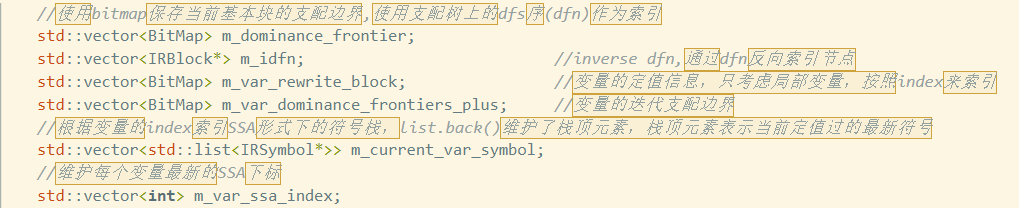
ConvertSSA.h

ConvertSSA.cpp

RevertSSA.h

RevertSSA.cpp

1. 数据结构



1. 功能说明
2. 普通形式IR转为SSA形式

类：ConvertSSA

函数：run

输入：普通形式的IRProgram

输出：SSA形式的IRProgram

算法步骤：

使用迭代支配边界法。



1. SSA形式转为原形式

类：RevertSSA

函数：run

输入：SSA形式的IRProgram

输出：普通形式的IRProgram

1. **常量折叠与代数化简**
2. 源文件

constant\_fold\_ast.cpp

constant\_fold\_ast.h

constant\_fold\_and\_algebraic\_simplification.cpp

constant\_fold\_and\_algebraic\_simplification.h

1. 数据结构

无

1. 功能说明
2. 对抽象语法树进行常量折叠

通过遍历抽象语法树将类型不是常量但却是字面值的结点变成常量结点

类：ConstantFoldAST

函数：fold

输入：AST

输出：AST

1. 对控制流图进行常量折叠和代数化简

通过遍历CFG对其进行常量折叠和代数化简

类：ConstantFoldAndAlgebraicSimplification

函数：control\_flow\_graph\_fold

输入：CFG

输出：CFG

1. **空基本块删除**
2. 源文件

empty\_block\_eliminate.h

empty\_block\_eliminate.cpp

1. 数据结构

无

1. 功能说明

（1）删除所有空的基本块

类：EmptyBlockEliminate

函数：run

输入：IRProgram

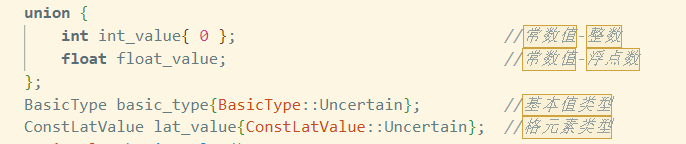
输出：删除所有空基本块后的IRProgram

1. **稀疏条件常量传播**
2. 源文件

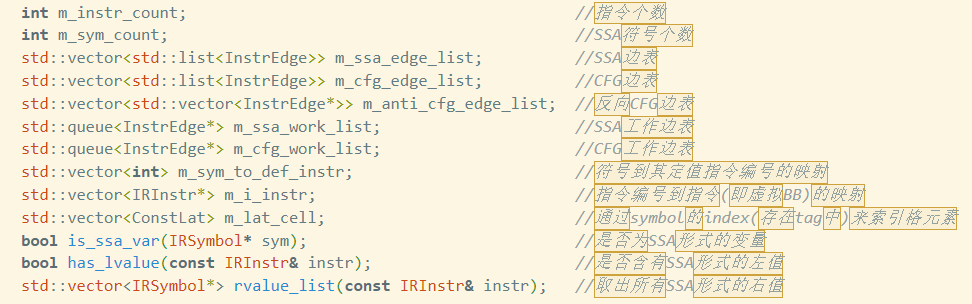
sparse\_conditional\_constant\_propagation.h

sparse\_conditional\_constant\_propagation.cpp

1. 数据结构
2. ConstLat



1. 其他数据



3、功能说明

（1）ConstLat

常量传播格。支持交运算，以及+-\*/%，类型转换，关系运算，取负的操作。

（2）稀疏条件常量传播

类：SparseConditionalConstantPropagation

函数：run

输入：SSA形式的IRProgram

输出：优化后的SSA形式的IRProgram

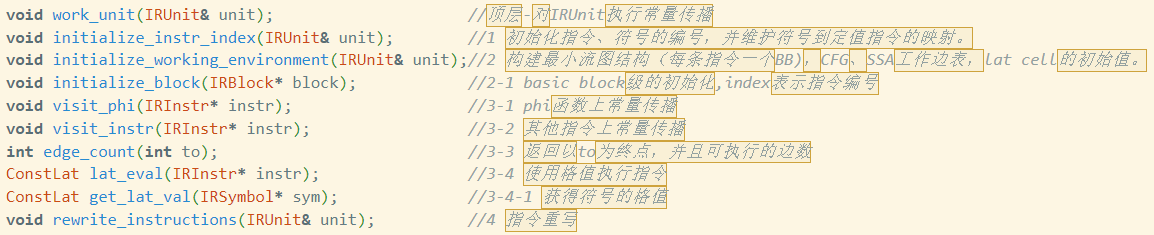
算法步骤：

①初始化，构建各种数据结构

②构建流图最小结构，初始化CFG、SSA工作边表以及lat cell。

②在流图边和SSA边上进行常量传播

③指令重写



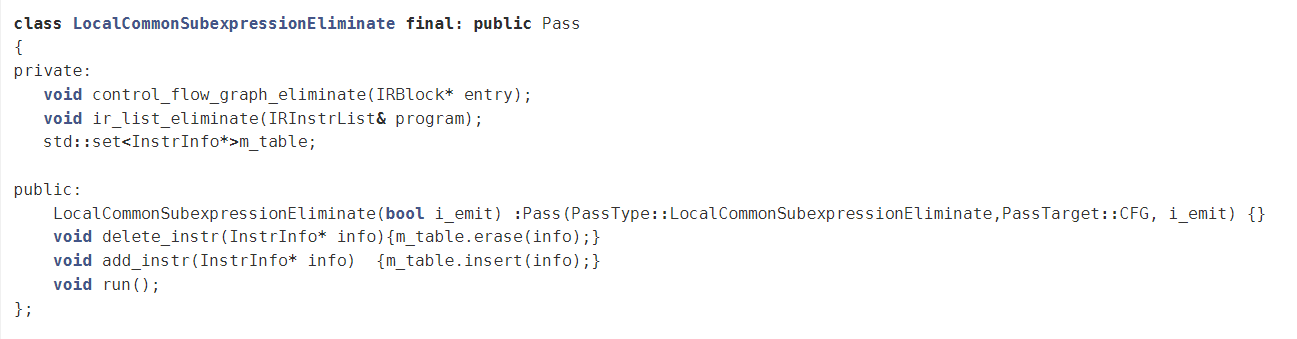
1. **公共子表达式删除**
2. 源文件

local\_common\_subexpression\_elimination.h

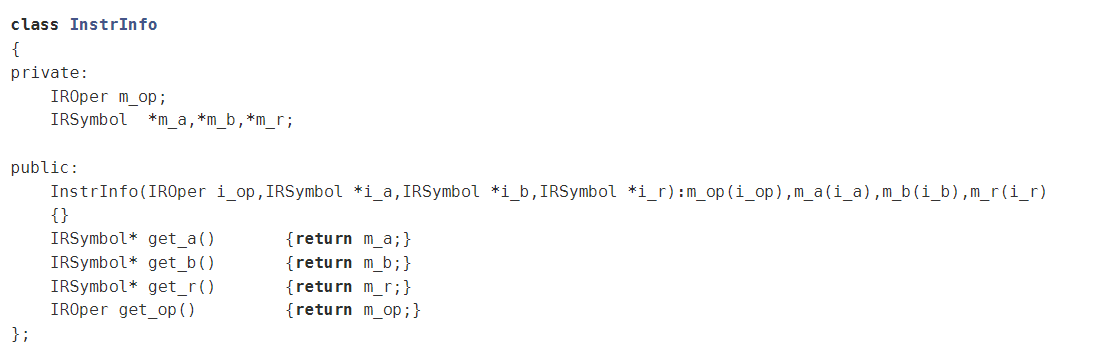
local\_common\_subexpression\_elimination.cpp

1. 数据结构

（1）LocalCommonSubexpressionEliminate



（2）InstrInfo



3、功能说明

（1）LocalCommonSubexpressionEliminate

公共子表示删除类继承基本优化类型，数据结构中主要是用于存储同一函数内部当前已经出现过表达式的表。

（2）InstrInfo

包含了三个操作数以及操作符。

1. 公共子表达式删除

搜索同一函数内部二元表达式并存储起来，当指令中出现了先前已经计算过的表达式便可替换，需要注意先前表达式中的操作数要保证未修改过。

类：LocalCommonSubexpressionEliminate

函数：run、ir\_list\_eliminate、control\_flow\_gragh\_eliminate

输入：程序控制流图

输出：程序控制流图

1. 增加表达式信息

将新的表达式信息加入全局表达式集合中

类：LocalCommonSubexpressionEliminate

函数：add\_instr

输入：InstrInfo类

输出：无

1. 删除表达式信息

从全局表达式集合中删除相关表达式

类：LocalCommonSubexpressionEliminate

函数：delete\_instr

输入：无

输出：无

1. **ARM汇编生成**
2. 源文件

arm.cpp

arm.h

arm\_manager.cpp

arm\_manager.h

1. 数据结构
2. Shift

ShiftOp op;

**bool** is\_imm;

**int** imm\_val;

    Reg\* reg;

1. Operand2

**bool** is\_imm8m;

**int** imm\_val;

    Reg\* reg;

    Shift\* shift;

1. ArmInstr

    Cond cond; *//用于条件执行，条件标记满足cond指示的条件时指令才执行*

    InstrType type; *//指令类型*

    BinaryOp op; *//用于二元运算*

**bool** s; *//s=1则更新条件标记*

**bool** is\_l; *//用于branch，is\_l=1表示链接*

**bool** is\_mvn;

    Reg\* rn; *//第一个源操作寄存器*

    Reg\* rd; *//目标寄存器*

    Operand2\* operand2; *//在load/store中作为reg\_offset*

    std::string label; *//用于branch指令*

**bool** is\_imm\_offset; *//用于load/store指令，表示偏移量为立即数*

**int** imm\_offset;

**int** imm; *//用于ldrpseudo伪指令*

**bool** is\_x; *//用于return，bx lr*

1. 功能说明
2. shift

作为Operand2的第二部分，即移位部分，可能是常数也可能是寄存器的值

1. Operand2

Operand2在指令中通常作为第二个操作数，具有多种形式：

* imm8m 在芯片处理时表示一个32位数，但是它是由一个8位数通过循环移位偶数位得到
* Reg {, <opsh>} (移动常数个位)，即寄存器的值移动常数个位后得到的值
* Reg, LSL/LSR/ASR/ROR Rs （寄存器经过移位后的值，移动位数为（自己或其它）寄存器的值）

1. ArmInstr

描述一条汇编指令，指令的每个成员可见2数据结构中的描述

1. arm汇编的生成

通过扫描CFG生成arm汇编

类：ArmManger

函数：gen\_arm

输入：控制流图，寄存器分配表

输出：arm汇编