MiniGo

Compiler for a subset of Golang

目录

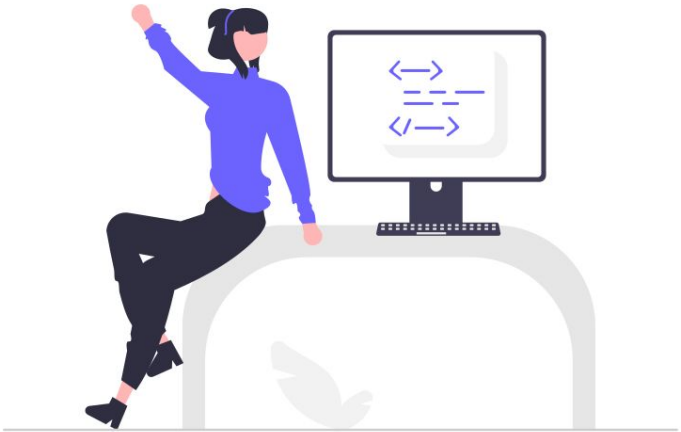
项 目 概 述

MiniGo

本项目实现了将 Golang 代码编译到 LLVM IR 再到 AMD64 机器代码。

前端使用C++实现，基于Flex&Bison。
后端使用Python实现。

支持大部分Golang基础语法，能够进行类型检查/推导和语法错误检查，进行了一些简单优化。



语 法 实 现

类型：int / multi-dimensional array

关键字：

```
"package"
"var"
"func"
"return"
"if"
"else"
"for"
"break"
"continue"
"defer"
"goto"
"const"
"nil"
```

对应特性：

1. 变量**声明**：`var a int`
2. 变量**初始化**：`var a, b int = 1, 2`

   `c, d := 3, 4`
3. **控制**结**构**：`if, else, break, continue`
4. **循**环结**构**：`for`
5. **函数**：`func add(a int, b int) {...}`
6. **多**维**数**组：`array := make([][]int, 4)`
7. **各种表达式**：`+ - * / %`

...

关于Statement：

```
Param : IDENT BType ;
ParamList : /* empty */  | Param  | ParamList ',' Param ;
FuncDef : FUNC IDENT '(' ParamList ')' ReturnType Block ;
ReturnType : /* empty */  | BType;
Block : '{}' ;
StmtList : /* empty */  | Stmt  | StmtList Stmt ;
ExpStmt : Exp ;
IncDecStmt : LVal INC  | LVal DEC ;
AssignStmt : LVals '=' InitVals  | LVal BIN_ASSIGN InitVal ;
ShortVarDecl : LVals DEFINE InitVals ;
SimpleStmt : /* empty stmt */
           | ExpStmt
           | IncDecStmt
           | AssignStmt
           | ShortVarDecl
           ;
IfStmt : IF Exp Block  | IF Exp Block ELSE Block  | IF Exp Block ELSE IfStmt  |
IF SimpleStmt ';' Exp Block  | IF SimpleStmt ';' Exp Block ELSE Block  | IF
SimpleStmt ';' Exp Block ELSE IfStmt ;
ReturnStmt : RETURN Exp  | RETURN ;
// BREAK, CONTINUE, GOTO
BranchStmt : BREAK  | CONTINUE  | GOTO IDENT ;
ForStmt : FOR Block  | FOR Exp Block  | FOR SimpleStmt ';' Exp ';' SimpleStmt
Block { // for
};
Stmt : Decl | IfStmt | ReturnStmt | SimpleStmt | ForStmt | Block | BranchStmt ;
```

EBNF

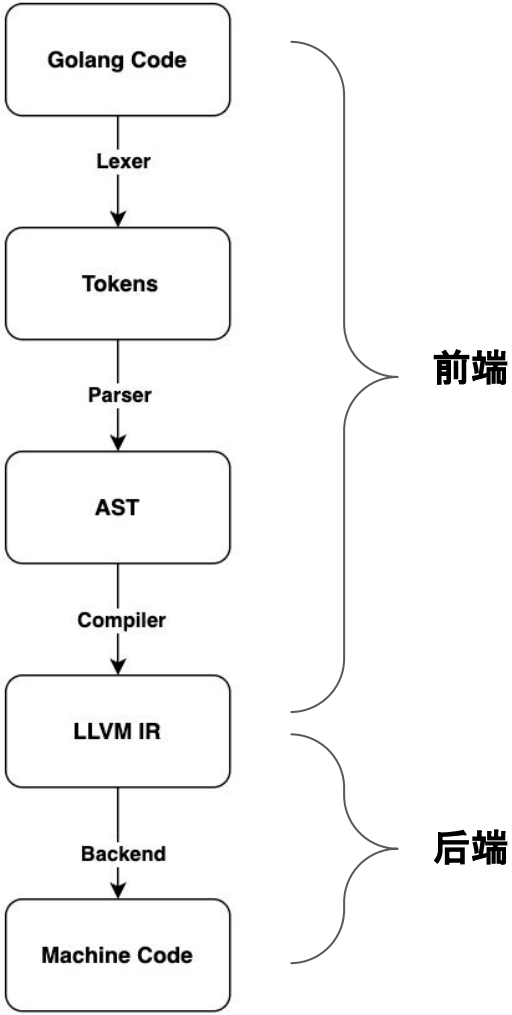
语法实现

关于变量&表达式：

```
// array: var a [2]type
IDs : IDENT  | IDs ',' IDENT ;
// initVal can be exp, array exp, make exp
InitVal : BType '{}'  | MAKE '(' BType ',' Exp ')'  | Exp;
InitVals : InitVal  | InitVals ',' InitVal ;
InitValList : /* empty */  | InitVals;
ConstInitVal  : ConstExp ;
ConstInitVals : ConstInitVal  | ConstInitVals ',' ConstInitVal ;

VarDecl : VAR VarSpec ;
VarSpec : IDs BType '=' InitVals | IDs '=' InitVals | IDs BType ;
ConstDecl : CONST ConstSpec ;
ConstSpec : IDs BType '=' ConstInitVals | IDs '=' ConstInitVals ;

Decl : VarDecl | ConstDecl;
ConstIndex : '[' ConstExp ']' | '[' ']' ;
ConstIndexList :  | ConstIndexList ConstIndex ;
BType : ConstIndexList INT ;

Number : INT_CONST | CHAR_CONST ;
Exp : LOrExp;
LVal : IDENT  | LVal '[' Exp ']' ;
LVals : LVal  | LVals ',' LVal ;
PrimaryExp : '(' Exp ')'  | LVal | Number  | NIL ;
UnaryExp : PrimaryExp | IDENT '(' ArgList ')'  | UnaryOp UnaryExp ;
UnaryOp: '+' | '-' | '!';
ArgList : /* empty */  | Exp  | ArgList ',' Exp ;
MulExp: UnaryExp | MulExp MulOp UnaryExp ;
MulOp: '*' | '/' | '%';
AddExp: MulExp | AddExp AddOp MulExp ;
AddOp: '+' | '-';
RelExp: AddExp | RelExp RelOp AddExp ;
RelOp: '<' | '>'
     | LE | GE;
EqExp: RelExp | EqExp EqOp RelExp ;
EqOp: EQ | NE;
LAndExp: EqExp | LAndExp AND EqExp ;
LOrExp: LAndExp | LOrExp OR LAndExp ;
ConstExp: Exp ;
```

模 块 介 绍

模块总览


Golang Code
Lexer
Tokens
Parser
AST
Compiler
LLVM IR
Backend
Machine Code
前端
后端

模块介绍

使用Flex工具完成，代码位于miniGo.l.

作用：用于将Golang源代码转化为Token流，忽略掉了注释以及空白字符等。

特性：包括了对字符字面量、八进制和十六进制字面量的支持。

```
/* 空白符和注释 */
WhiteSpace    [ \t\n\r]*
LineComment   "//".*

/* char literal */
SimpleEscapeSequence \\'|\\"|\\?|\\\\|\a|\b|\f|\n|\r|\t|\v
CharLiteral   \'[^\\'\n]\'|\'{SimpleEscapeSequence}\'

/* 标识符 */
Identifier    [a-zA-Z_][a-zA-Z0-9_]*

/* 整数字面量 */
Decimal       [1-9][0-9]*
Octal         0[0-7]*
Hexadecimal   0[xX][0-9a-fA-F]+

%%

{WhiteSpace}    { /* 忽略，不做任何操作 */ }
{LineComment}   { /* 忽略，不做任何操作 */ }

{CharLiteral}   { yylval.char_val = getCharVal(yytext); return CHAR_CONST; }
```

```
"package"       { return PACKAGE; }
"import"        { return IMPORT; }
"var"           { return VAR; }
"func"          { return FUNC; }
"return"        { return RETURN; }
"if"            { return IF; }
"else"          { return ELSE; }
"for"           { return FOR; }
"break"         { return BREAK; }
"continue"      { return CONTINUE; }
"defer"         { return DEFER; }
"goto"          { return GOTO; }
"const"         { return CONST; }
"make"          { return MAKE; }
":="            { return DEFINE; }
"++"            { return INC; }
"--"            { return DEC; }

"int"           { return INT; }
"nil"           { return NIL; }

"=="            { yylval.str_val = new string(yytext); return EQ; }
"!="            { yylval.str_val = new string(yytext); return NE; }
"<="            { yylval.str_val = new string(yytext); return LE; }
">="            { yylval.str_val = new string(yytext); return GE; }
"&&"            { yylval.str_val = new string(yytext); return AND; }
"||"            { yylval.str_val = new string(yytext); return OR; }

"+="            { yylval.str_val = new string(yytext); return BIN_ASSIGN; }
"-="            { yylval.str_val = new string(yytext); return BIN_ASSIGN; }
"*="            { yylval.str_val = new string(yytext); return BIN_ASSIGN; }
"/="            { yylval.str_val = new string(yytext); return BIN_ASSIGN; }
"%="            { yylval.str_val = new string(yytext); return BIN_ASSIGN; }

{Identifier}    { yylval.str_val = new string(yytext); return IDENT; }

{Decimal}       { yylval.int_val = strtol(yytext, nullptr, 0); return INT_CONST; }
{Octal}         { yylval.int_val = strtol(yytext, nullptr, 0); return INT_CONST; }
{Hexadecimal}   { yylval.int_val = strtol(yytext, nullptr, 0); return INT_CONST; }

.               { yylval.char_val = yytext[0]; return yytext[0]; }
```

Parser

使用Yacc工具完成，相关代码包括miniGo.y和AST.hpp.

作用：由Token流构建AST。

特性：通过转换为Equivalent AST的方式来支持一些语法糖，例如 a+=1 -> a=a+1。

支持以Json格式输出AST。

使用方法：

```
unique_ptr<BaseAST> ast;
auto ret = yyparse(ast);
assert(!ret);
// To get a json of AST
FILE *astFp = fopen("ast.o.json", "w");
fprintf(astFp, "%s", ast->toJson().dump(4).c_str());
fclose(astFp);
```

Parser

使用Yacc工具完成，相关代码包括miniGo.y和AST.hpp.

作用：由Token流构建AST。

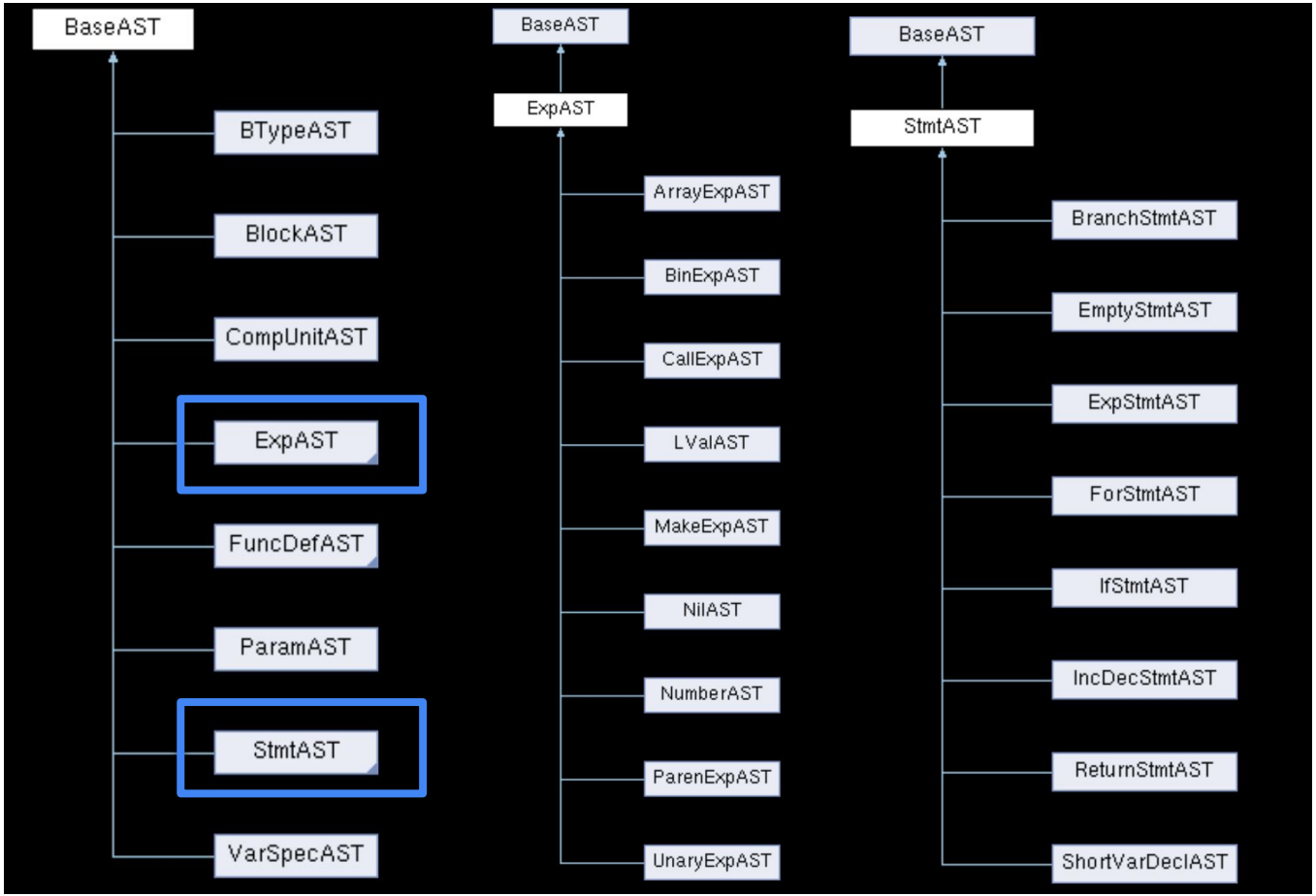
特性：通过转换为Equivalent AST的方式来支持一些语法糖，例如 a+

支持以Json格式输出AST。

使用方法：

```
unique_ptr<BaseAST> ast;
auto ret = yyparse(ast);
assert(!ret);
// To get a json of AST
FILE *astFp = fopen("ast.o.json", "w");
fprintf(astFp, "%s", ast->toJson().dump(4).c_str());
fclose(astFp);
```

Parser - AST


BaseAST
BTypeAST
BlockAST
CompUnitAST
ExpAST
FuncDefAST
ParamAST
StmtAST
VarSpecAST
BaseAST
ExpAST
ArrayExpAST
BinExpAST
CallExpAST
LValAST
MakeExpAST
NilAST
NumberAST
ParenExpAST
UnaryExpAST
BaseAST
StmtAST
BranchStmtAST
EmptyStmtAST
ExpStmtAST
ForStmtAST
IfStmtAST
IncDecStmtAST
ReturnStmtAST
ShortVarDeclAST

Public Member Functions

virtual	**~BaseAST** ()=default Destroy the **BaseAST** object.
virtual TType	**type** () const =0 get the type of the AST
virtual json	**toJson** () const =0 get the json representation of the AST
void	**setParent** (**BaseAST** ***parent**) Set the Parent object.
BaseAST *	**getParent** () Get the Parent object.
virtual void	**add** (**BaseAST** *ast) add a child to the AST
virtual string	**info** () const get the type info of the AST
virtual **BaseAST** *	**copy** () const copy the AST itself

相关代码包括 `Compiler.hpp`, `AST.hpp` and `Scope.hpp`.

作用：由AST生成对应的LLVM IR。

特性：

1. 类型推导，详见下面的Type inference章节。

2. 类型检查，详见下面的Type checking章节。

3. 语法报错。

Compiler

模块介绍

	Compiler () Construct a new **Compiler** object with the Universe scope.
string	**Compile** (**CompUnitAST** *_file) compile the whole **CompUnitAST** (the file)
void	**enterScope** () enter a new scope
void	**leaveScope** () leave the current scope to the outer scope
void	**restoreScope** (**Scope** *s) restore the scope to the given scope
void	**genHeader** (ostream &os, **CompUnitAST** *file) generate the header, including the package name and the runtime functions
void	**genMain** (ostream &os, **CompUnitAST** *file) generate the main function, which calls main_init and main_main
string	**genId** () generate a new temp's id of LLVM IR
string	**genLabelId** (string name) generate a new label with unique id
void	**genDefaultInit** (ostream &os, string varType, string varMName) generate the LLVM IR of init a var with default val (usually zero val)
void	**genInit** (ostream &os, **CompUnitAST** *file) generate the function to init global vars called X_init
void	**compileFile** (ostream &os, **CompUnitAST** *file) compile the file without the header and fake main function
void	**compileFunc** (ostream &os, **CompUnitAST** *file, **FuncDefAST** *fn) compile a function
void	**compileStmt** (ostream &os, const pAST &_stmt) compile a statement
void	**compileStmt_assign** (ostream &os, const pAST &_stmt) compile a assign statement
string	**compileExpr** (ostream &os, const pAST &_expr) compile a expression
string	**reduceDim** (string t)

Compiler - 类型推导

Type inference is needed because of statements like `**i := arr[1]**`.

Some simple type inference is done by `**Compiler::inferType(pAST exp)**` and `**BaseAST::info()**`.

```
string inferType(const pAST& _expr) {
    auto expr = reinterpret_cast<ExpAST*>(_expr.get());
    if (expr->type() == TType::NumberT) {···
    } else if (expr->type() == TType::BinExpT) {
        auto exp = reinterpret_cast<BinExpAST*>(expr);
        auto left = inferType(exp->left);
        auto right = inferType(exp->right);
        if (left != right) {
            cerr << "inferType: type mismatch" << endl;
            assert(false);
        }
        return left;
    } else if (expr->type() == TType::UnaryExpT) {···
    } else if (expr->type() == TType::ParenExpT) {···
    } else if (expr->type() == TType::CallExpT) {···
    } else if (expr->type() == TType::LValT) {
        auto exp = reinterpret_cast<LValAST*>(expr);
        auto obj = scope->Lookup(exp->ident).second;
        if (obj == nullptr) {
            cerr << "inferType: variable " << exp->ident << " undefined"
                 << endl;
            assert(false);
        }
        auto baseType = obj->Node->info();
        if (exp->indexList == nullptr) {
            cerr << "inferType: indexList is null" << endl;
            assert(false);
        }
        return reduceDim(baseType, exp->indexList->size());
    } else if (expr->type() == TType::MakeExpT) {···
    } else if (expr->type() == TType::ArrayExpT) {···
    } else if (expr->type() == TType::NilT) {···
    } else {···
    return "";
}
```

Compiler - 类型检查


```
} else if (expr→type() == TType::CallExpT) {
    auto exp = reinterpret_cast<CallExpAST*>(expr);
    auto obj = scope→Lookup(exp→funcName).second;
    string funcName;
    if (obj ≠ nullptr) {···
    } else {···
    auto funcDefAST = reinterpret_cast<FuncDefAST*>(obj→Node);
    int argNum = exp→argList→size();
    // get return type and param types
    auto paramTypes = funcDefAST→getParamTypes();
    string funcType = funcDefAST→info();
    // args
    vector<string> argNames;
    vector<string> argTypes;
    for (int i = 0; i < argNum; i++) {···
    // no localName if return void
    auto returnType = funcDefAST→getRetType();
    stringstream sub;
    if (returnType == "void") {···
    } else {···
    os << "\t" << sub.str() << "call " << funcType << " " << funcName
       << "(";
    for (int i = 0; i < argNum; i++) {
        // type check for param and arg
        if (!typeMatch(paramTypes→at(i), argTypes[i])) {
            cerr << "compileExpr: type mismatch in function call - "
                 << funcName << endl;
            // show the arg type and param type
            cerr << " - arg has type \"" << argTypes[i]
                 << "\", but expected \"" << paramTypes→at(i) << "\"."
                 << endl;
            cerr << " - arg AST: " << *exp→argList→at(i) << endl;
            assert(false);
        }
        os << paramTypes→at(i) << " " << argNames[i];
        if (i ≠ argNum - 1) os << ", ";
    }
    os << ")" << endl;
    return localName;
} else {···
```

以编译CallExpression为例：

检查每一个参数的类型是否符合函数定义。

Backend

相关代码包括 `Backend.py`.

作用：**由LLVM IR生成对应的AMD64机器码。**

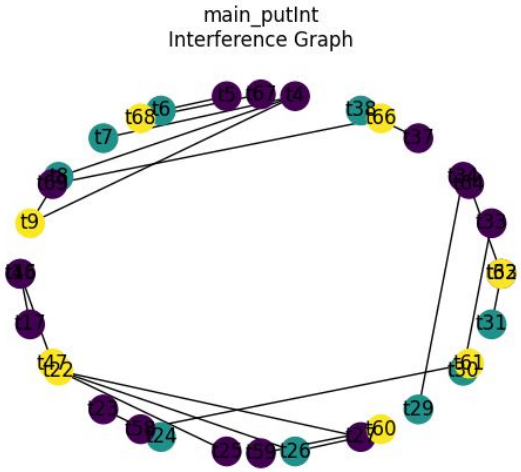
特性：

1. 每个函数单独处理，推导变量活性，生成冲突图。

：只使用R8-R15作为临时变量(足够)

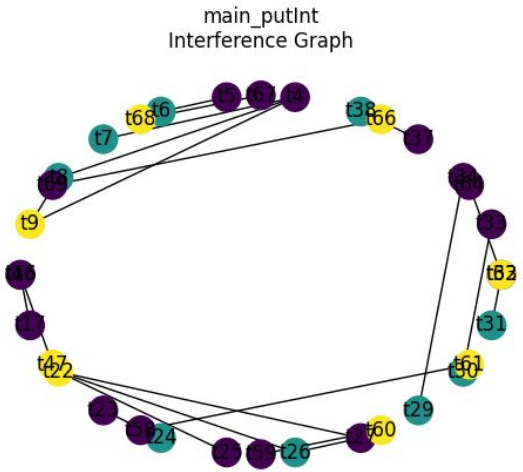
2. 传参使用RDI、RSI、RDX、RCX，其余通过压栈传入。

3. 部分简单优化。



```
define void @main_putInt(i64 %local_n.26.arg0) {
    %local_n.26 = alloca i64, align 4
    store i64 %local_n.26.arg0, i64* %local_n.26
    %t243 = load i64, i64* %local_n.26, align 4
    %t244 = add i64 0, 0
    %t242 = icmp slt i64 %t243, %t244
    br i1 %t242, label %if.body14.239, label %if.end14.241

if.body14.239:
    %t245 = add i64 0, 45
    %t246 = call i64(i64) @putchar(i64 %t245)
    %t248 = load i64, i64* %local_n.26, align 4
    %t247 = sub i64 0, %t248
    store i64 %t247, i64* %local_n.26
    br label %if.end14.241

if.end14.241:
    %t256 = load i64, i64* %local_n.26, align 4
    %t257 = add i64 0, 10
    %t255 = sdiv i64 %t256, %t257
    %t258 = add i64 0, 0
    %t254 = icmp ne i64 %t255, %t258
    br i1 %t254, label %if.body15.251, label %if.end15.253

if.body15.251:
    %t260 = load i64, i64* %local_n.26, align 4
    %t261 = add i64 0, 10
    %t259 = sdiv i64 %t260, %t261
    call void(i64) @main_putInt(i64 %t259)
    br label %if.end15.253

if.end15.253:
    %t264 = load i64, i64* %local_n.26, align 4
    %t265 = add i64 0, 10
    %t263 = srem i64 %t264, %t265
    %t266 = add i64 0, 48
    %t262 = add i64 %t263, %t266
    %t267 = call i64(i64) @putchar(i64 %t262)
    ret void
}
```

```
// fmt.Printf("%d", n)
func putInt(n int) {
    if n < 0 {
        putchar('-')
        n = -n
    }
    if n/10 != 0 {
        putInt(n / 10)
    }
    putchar(n%10 + '0')
}
```



```
define void @main_putInt(i64 %local_n.26.arg0) {
    %local_n.26 = alloca i64, align 4
    store i64 %local_n.26.arg0, i64* %local_n.26
    %t243 = load i64, i64* %local_n.26, align 4
    %t244 = add i64 0, 0
    %t242 = icmp slt i64 %t243, %t244
    br i1 %t242, label %if.body14.239, label %if.end14.241

if.body14.239:
    %t245 = add i64 0, 45
    %t246 = call i64(i64) @putchar(i64 %t245)
    %t248 = load i64, i64* %local_n.26, align 4
    %t247 = sub i64 0, %t248
    store i64 %t247, i64* %local_n.26
    br label %if.end14.241

if.end14.241:
    %t256 = load i64, i64* %local_n.26, align 4
    %t257 = add i64 0, 10
    %t255 = sdiv i64 %t256, %t257
    %t258 = add i64 0, 0
    %t254 = icmp ne i64 %t255, %t258
    br i1 %t254, label %if.body15.251, label %if.end15.253

if.body15.251:
    %t260 = load i64, i64* %local_n.26, align 4
    %t261 = add i64 0, 10
    %t259 = sdiv i64 %t260, %t261
    call void(i64) @main_putInt(i64 %t259)
    br label %if.end15.253

if.end15.253:
    %t264 = load i64, i64* %local_n.26, align 4
    %t265 = add i64 0, 10
    %t263 = srem i64 %t264, %t265
    %t266 = add i64 0, 48
    %t262 = add i64 %t263, %t266
    %t267 = call i64(i64) @putchar(i64 %t262)
    ret void
}
```

```
    .globl main_putInt
    main_putInt:
pushq %rbp
movq %rsp, %rbp
subq $8, %rsp
pushq %r8
pushq %r9
  #  store i64 %local_n.26.arg0, i64* %local_n.26, align 4
movq %rdi, -8(%rbp)
  #  %t243 = load i64, i64* %local_n.26, align 4
movq -8(%rbp), %r8
  #  %t244 = add i64 0, 0
movq $0, %r9
  #  %t242 = icmp slt i64 %t243, %t244
movq $0, %rax
cmpq %r9, %r8
setl %al
movq %rax, %r8
  #  br i1 %t242, label %if.body14.239, label %if.end14.241
cmpq $0, %r8
je if.end14.241
    if.body14.239:
  #  %t245 = add i64 0, 45
movq $45, %r8
  #  %t246 = call i64 @putchar(i64 %t245)
movq %r8, %rdi
call putchar@PLT
movq %rax, %r8
  #  %t248 = load i64, i64* %local_n.26, align 4
movq -8(%rbp), %r8
  #  %t247 = sub i64 0, %t248
movq %r8, %rax
movq $0, %r8
subq %rax, %r8
  #  store i64 %t247, i64* %local_n.26, align 4
movq %r8, -8(%rbp)
```

```
if op == "add":
    rs1 = toR(regs[0])
    rs2 = toR(regs[1])
    rd = toR(i)
    # do not overwrite rs2
    if rd == rs2:
        rs1, rs2 = rs2, rs1
    cmds += [f"movq {rs1}, {rd}"]
    cmds += [f"addq {rs2}, {rd}"]
    if optimize:
        isI = isImm(rs1, rs2)
        # both are imm
        if None not in isI:
            res = isI[0] + isI[1]
            cmds = [f"movq ${res}, {rd}"]
```

如果两个操作数都是立即数，

则直接把结果mov到目标寄存器。

```
if optimize:
    # remove useless movq
    rm = []
    for line in flat.split('\n'):
        if line.startswith('movq'):
            if line.split()[1][:-1] == line.split()[2]:
                rm += [line]
    for line in rm:
        flat = flat.replace(line+'\n', '')
    # remove the second br in two consecutive br
    # no label in between of course
    rm = []
    lines = flat.split('\n')
    for idx, line in enumerate(lines):
        if line.startswith('br'):
            if lines[idx-1].startswith('br'):
                rm += [line]
    for line in rm:
        flat = flat.replace(line+'\n', '')
```

对最后的AMD64汇编代码：

1. 去除所有无用的mov
2. 去除连续两个的br中的后者

经过测试，

各个简单优化后汇编代码减少约5%。

系 统 演 示

通过测试

系统演示

```
❯ make mini_build
mkdir -p build
flex -o build/miniGo.yy.cpp src/miniGo.l
bison -t src/miniGo.y -o build/miniGo.tab.hpp
src/miniGo.y: warning: 51 shift/reduce conflicts [-Wconflicts-sr]
src/miniGo.y: warning: 18 reduce/reduce conflicts [-Wconflicts-rr]
src/miniGo.y: note: rerun with option '-Wcounterexamples' to generate conflict counterexamples
--- Build Compiler ---
clang++ -o build/miniGo build/miniGo.yy.cpp src/main.cpp -std=c++20 -Isrc -Ibuild
--- Build LL ---
build/miniGo tests/course_selection.go -o build/course_selection.o.ll
>> parsing ...
>> done
--- Build Bin ---
clang build/course_selection.o.ll -o build/course_selection.bin
--- Build LL ---
build/miniGo tests/matrix.go -o build/matrix.o.ll
>> parsing ...
>> done
--- Build Bin ---
clang build/matrix.o.ll -o build/matrix.bin
--- Build LL ---
build/miniGo tests/sort.go -o build/sort.o.ll
>> parsing ...
>> done
--- Build Bin ---
clang build/sort.o.ll -o build/sort.bin
rm build/course_selection.o.ll build/sort.o.ll build/matrix.o.ll
❯ ./testers/quicksort/quicksort-linux-amd64 build/sort.bin
fixed case 0 (size 0) ... pass!
fixed case 1 (size 1) ... pass!
fixed case 2 (size 2) ... pass!
fixed case 3 (size 2) ... pass!
fixed case 4 (size 3) ... pass!
fixed case 5 (size 3) ... pass!
fixed case 6 (size 3) ... pass!
fixed case 7 (size 3) ... pass!
fixed case 8 (size 3) ... pass!
fixed case 9 (size 4) ... pass!
fixed case 10 (size 9) ... pass!
fixed case 11 (size 9) ... pass!
fixed case 12 (size 10000) ... pass!
fixed case 13 (size 10000) ... pass!
fixed case 14 (size 4096) ... pass!
randomly generated case 0 (size 10000) ... pass!
randomly generated case 1 (size 10000) ... pass!
randomly generated case 2 (size 10000) ... pass!
randomly generated case 3 (size 10000) ... pass!
randomly generated case 4 (size 10000) ... pass!
randomly generated case 5 (size 10000) ... pass!
```

恳 请 批 评 指 正