Lab2 实验报告

PB20111633
阿非提

实验要求

- **LLVM IR**
  - 手动编写 .ll
  - 完成 `./tests/2-ir-gen-warmup/stu_ll` 目录下的4个文件
  - 回答问题1: getelementptr
- **LightIR**
  - 利用 LightIR + cpp 编写生成 .ll 的程序
  - 完成 `./tests/2-ir-gen-warmup/stu_cpp` 目录下的4个文件
  - 回答问题2: cpp 与 .ll 的对应
- **Lab3 的准备**
  - 回答问题3: Visitor Pattern

三个问题

问题1: getelementptr

getelementptr 指令
功能：用于地址计算，并不进行内存访问。
参数：
第一： 计算基础类型。
第二： 基地址。
其余： 所要计算地址的元素在其聚合对象中的索引。

```
%2 = getelementptr [10 x i32], [10 x i32]* %1, i32 0, i32 %0
```

此时计算需要一个
参数一：`[10 x i32]`，计算基础类型为大小为10的32位整型数组
参数二：`[10 x i32]* %1`，基地址为 `%1` 的值
参数三：`i32 0`，表示从地址 `%1` 开始的第一个这样的数组
参数四：`i32 %0`，参数三指向的数组中的第 `%0+1` 个元素

```
%2 = getelementptr i32, i32* %1, i32 %0
```

此时计算需要一个
参数一：`i32`，计算基础类型为32位整型
参数二：`i32* %1`，基地址为 `%1` 的值
参数三：`i32 %0`，表示从地址 `%1` 开始的第 `%0` 个这样的32位整形

问题2: cpp 与 .ll 的对应

以下使用表格的方式做出了cpp文件和 .ll文件中代码的对应关系（即 cpp 列表格每行代码生成 .ll 列表格中对应行中的代码。

assign.c

cpp	.ll
auto bb = BasicBlock::create(module, "entry", Function::create(FunctionType::get(Int32Type, {}), "main", module));	
builder->set_insert_point(bb);	label_entry:
auto array_a = builder->create_alloca(ArrayType::get(Int32Type, 10));	%op0 = alloca [10 x i32]
auto a0 = builder->create_gep(array_a, {ConstantInt::get(0, module), ConstantInt::get(0, module)});	%op1 = getelementptr [10 x i32], [10 x i32]* %op0, i32 0, i32 0
builder->create_store(ConstantInt::get(10, module), a0);	store i32 10, i32* %op1
auto load_a0 = builder->create_load(a0);	%op2 = load i32, i32* %op1
auto a0_mul_2 = builder->create_imul(load_a0, ConstantInt::get(2, module));	%op3 = mul i32 %op2, 2
auto a1 = builder->create_gep(array_a, {ConstantInt::get(0, module), ConstantInt::get(1, module)});	%op4 = getelementptr [10 x i32], [10 x i32]* %op0, i32 0, i32 1
builder->create_store(a0_mul_2, a1);	store i32 %op3, i32* %op4
auto load_a1 = builder->create_load(a1);	%op5 = load i32, i32* %op4
builder->create_ret(load_a1);	ret i32 %op5

fun.c

cpp	.ll

auto bb = BasicBlock::create(module, "entry", calleeFunc);
 	
builder->set_insert_point(bb);
 	
auto a = builder->create_alloca(Int32Type);
 	
auto arg = calleeFunc->arg_begin();
 	
builder->create_store(*arg, a);
 	
auto load_a = builder->create_load(a);
 	
auto mul = builder->create_imul(ConstantInt::get(2, module), load_a);
 	
builder->create_ret(mul);
	label_entry:
	
	%op1 = alloca i32
	
	store i32 %arg0, i32* %op1
	
	%op2 = load i32, i32* %op1
	
	%op3 = mul i32 2, %op2
	
	ret i32 %op3

bb = BasicBlock::create(module, "entry", mainFunc);
 	
builder->set_insert_point(bb);
 	
auto call = builder->create_call(calleeFunc, {ConstantInt::get(110, module)});
 	
builder->create_ret(call);
	label_entry:
	
	%op0 = call i32 @callee(i32 110)
	
	ret i32 %op0

if.c

cpp	.ll

auto bb = BasicBlock::create(module, "entry", mainFunc);
 	
builder->set_insert_point(bb);
	label_entry:

auto a = builder->create_alloca(FloatType);	%op0 = alloca float
auto return_val = builder->create_alloca(Int32Type);	%op1 = alloca i32
builder->create_store(ConstantInt::get(0, module), return_val);	store i32 0, i32* %op1
builder->create_store(ConstantFP::get(5.555, module), a);	store float 0x40163851e0000000, float* %op0
auto load_a = builder->create_load(a);	%op2 = load float, float* %op0

auto fcmp = builder->create_fcmp_gt(load_a, ConstantFP::get(1.0, module));
 	
auto True = BasicBlock::create(module, "True", mainFunc);
 	
auto Return = BasicBlock::create(module, "Return", mainFunc);
	%op3 = fcmp ugt float %op2,0x3ff0000000000000

auto br = builder->create_cond_br(fcmp, True, Return);	br i1 %op3, label %label_True, label %label_Return

builder->set_insert_point(True);
 	
builder->create_store(ConstantInt::get(233, module), return_val);
	label_True:
	
	store i32 233, i32* %op1

builder->create_br(Return);	br label %label_Return

builder->set_insert_point(Return);
 	
auto load_return_val = builder->
create_load(return_val);
 	
builder->create_ret(load_return_val);
	label_Return:
	
	%op4 = load i32, i32* %op1
	
	ret i32 %op4

while.c

cpp	.ll

auto bb = BasicBlock::create(module,
"entry", mainFunc);
 	
builder->set_insert_point(bb);
 	
auto a = builder->
create_alloca(Int32Type);
 	
auto i = builder->
create_alloca(Int32Type);
 	
builder->
create_store(ConstantInt::get(10,
module), a);
 	
builder->
create_store(ConstantInt::get(0,
module), i);
 	
auto While =
BasicBlock::create(module, "While",
mainFunc);
 	
auto WhileLoop =
BasicBlock::create(module,
"WhileLoop", mainFunc);
 	
auto Return =
BasicBlock::create(module, "Return",
mainFunc);
 	
builder->create_br(While);
	label_entry:
	
	%op0 = alloca i32
	
	%op1 = alloca i32
	
	store i32 10, i32* %op0
	
	store i32 0, i32* %op1
	
	br label %label_While

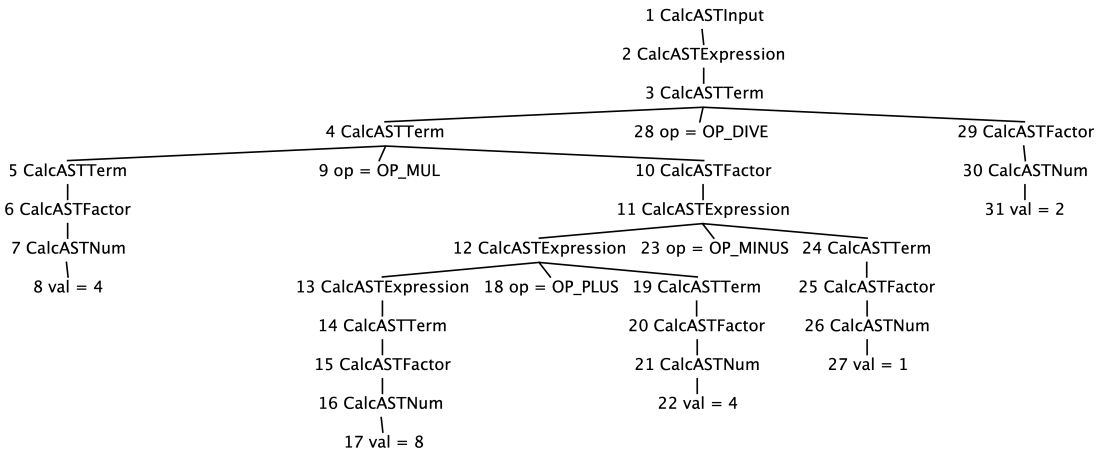
builder->set_insert_point(While);
 	
auto load_i = builder->create_load(i);
 	
auto icmp = builder->
create_icmp_lt(load_i,
ConstantInt::get(10, module));
 	
builder->create_cond_br(icmp,
WhileLoop, Return);
	label_While:
	
	%op2 = load i32, i32* %op1
	
	%op3 = icmp slt i32 %op2, 10
	
	br i1 %op3, label %label_WhileLoop,
	label %label_Return

builder->set_insert_point(WhileLoop);
 	
auto i_increment = builder->
create_iadd(load_i,
ConstantInt::get(1, module));
 	
builder->create_store(i_increment, i);
 	
auto load_a = builder->create_load(a);
 	
auto a_add_i = builder->
create_iadd(load_a, i_increment);
 	
builder->create_store(a_add_i, a);
 	
builder->create_br(While);
	label_WhileLoop:
	
	%op4 = add i32 %op2, 1
	
	store i32 %op4, i32* %op1
	
	%op5 = load i32, i32* %op0
	
	%op6 = add i32 %op5, %op4
	
	store i32 %op6, i32* %op0
	
	br label %label_While

builder->set_insert_point(Return);
 	
load_a = builder->create_load(a);
 	
builder->create_ret(load_a);
	label_Return:
	
	%op7 = load i32, i32* %op0
	
	ret i32 %op7

问题3: Visitor Pattern

1.



2. 1->2->3->4->5->6->7->8->9->10->11->12->13->14->15->16->17->18->19->20->21->22->23->24->25->26->27->28->29->30->31

实验难点

- 理解LLVM IR 中 getelementptr 指令的参数的作用

实验反馈

实验文档提供了所有实验所需的知识基础，节省了同学们自行查找资料的过程。整体实验体验很好。