

Гибкий подход к подъёму LLVM MIR кода в SSA форму LLVM IR

Выполнил: Романов Александр Викторович
Научный руководитель: Владимиров Константин Игоревич

Цели и Задачи

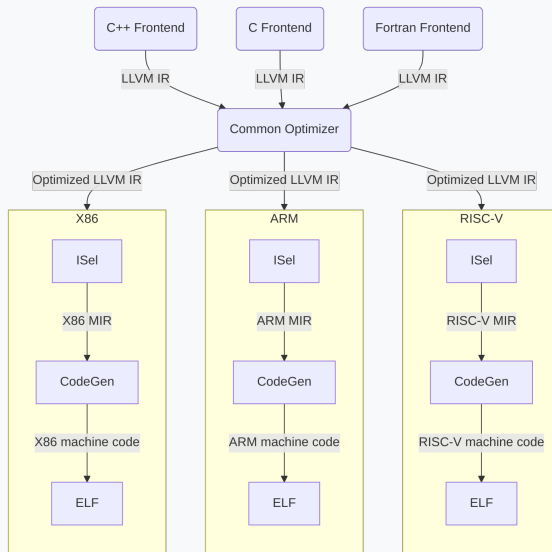
■ Цель

- Разработать инструмент для восстановления LLVM IR из машинного кода RISC-V

■ Задачи

- Изучить существующие подходы к подъёму машинного кода в высокоуровневое представление
- Разработать модель для семантического переноса RISC-V кода для RV64IM
- Разработать инструмент переноса кода на основе инфраструктуры LLVM

Компилятор LLVM



Уровни IR

■ High Level IR (HIR)

- Высокоуровневое представление
- Не зависит от языка программирования
- Не зависит от архитектуры
- Виртуальные инструкции и регистры
- Машинно-независимые оптимизации
- LLVM IR

■ Low Level IR (LIR)

- Низкоуровневое представление
- Специфично для архитектуры
- Физические регистры
- Конкретные инструкции целевой архитектуры
- Оптимизации машинного уровня
- LLVM MIR

Бинарные трансляторы

■ Обычные

- Специализированное низкоуровневое представление
- Сложная поддержка новых целевых архитектур
- Сопоставление регистров
- Сопоставление инструкций

■ Примеры

- Rosetta
- QEMU
- Berberis

■ LLVM-based

- Подъём в LLVM IR
- Развитая инфраструктура анализа и инструментации

■ Примеры

- mcsema
- mctoll
- instrew
- rellume - единственный поддерживает RISC-V (197 инструкций)

Общие принципы

■ Каждой инструкции сопоставляется функция

- Входной операнд -> аргумент
- Результат -> возвращаемое значение

■ Инструкция

```
$x1 = ADD $x2, $x3
```

■ Функция

```
define i64 @ADD(i64 %0, i64 %1) {  
    %3 = add i64 %1, %0  
    ret i64 %3  
}
```

Преобразование кода

Блок MIR (RISC-V)

```
bb.1:  
  $x19 = MUL $x7, $x2  
  $x28 = ADD $x21, $x19  
  $x13 = ADD $x13, $x1  
  BNE $x13, $x3, %bb.1
```

Получившийся LLVM IR

```
bb1:  
%19 = call i64 @MUL(i64 %x7, i64 %x2)  
%x28 = call i64 @ADD(i64 %x21, i64 %19)  
%13 = call i64 @ADD(i64 %x13, i64 %x1)  
%cmp = icmp ne i64 %13, %x3  
br i1 %cmp, label %bb1, label %bb2
```

LLVM IR после подстановки функций

```
bb1:  
%19 = mul i64 %x7, %x2  
%x28 = add i64 %x21, %19  
%13 = add i64 %x13, %x1  
%cmp = icmp ne i64 %13, %x3  
br i1 %cmp, label %bb1, label %bb2
```

Преобразование функций

■ Собственное соглашение о вызовах

Структура состояния - единственный аргумент

```
1 define void @foo(ptr %0) {  
2   %GPR = getelementptr %register_state, ptr %0, i32 0, i32 0  
3   ... # loading registers from state  
4   %x2_upd = add i64 %x2, -1  
5   ... # save updated registers to state  
6   call void @bar(ptr %0)  
7   ... # reloading registers from state  
8   %x23_upd = mul i64 %x16, %x14  
9   ... # save updated registers to state  
10  ret void  
11 }
```

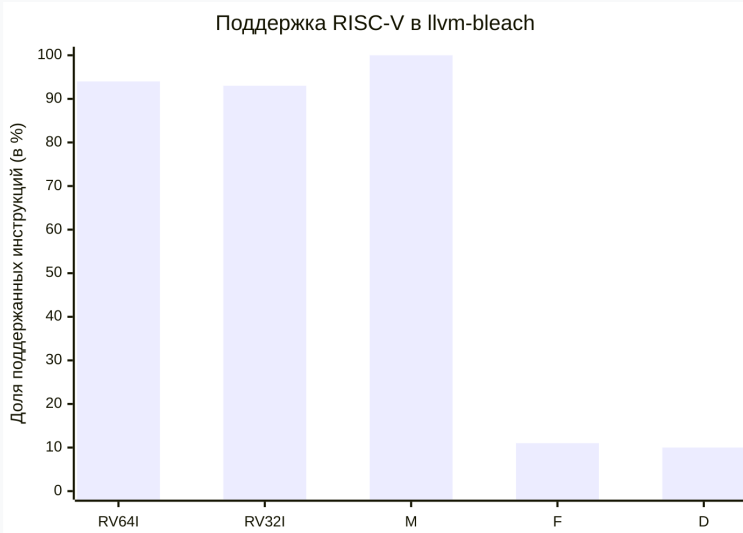

Алгоритм

```
1 def lift(F: function):
2   loadRegsFromState(F.StateArgument)
3   for MBB in F:
4     for I in MBB:
5       if I.isCall():
6         saveRegsToState(F.StateArgument)
7         BB.insertCall(I.Callee, F.StateArgument)
8         reloadRegs(F.StateArgument)
9       else if I.isCondBranch():
10        Cond = BB.insertCall(I.function, I.Src1, I.Src2)
11        BB.insertBranch(Cond, I.ifTrue, I.ifFalse);
12      else:
13        Regs[I.Dst] = BB.insertCall(I.func, I.Src1, I.Src2)
14    saveRegsToState(F.StateArgument)
```

Результаты

Поддержанные инструкции

- RV64I/E: 50 (94%)
- RV32I/E: 42 (93%)
- M Ext: 12 (100%)
- F Ext: 4 (11%)
- D Ext: 4 (10%)



Спасибо за внимание!