Санкт-Петербургский государственный университет
Кафедра системного программирования

# Динамическая бинарная трансляция в RISC-V с помощью Instrew

Михайлов Илья Игоревич

28.11.2023

**Научный руководитель:** ассистент кафедры системного программирования, Косарев Д. С.

Санкт-Петербург
2024

# Введение

- Бинарные трансляторы являются популярными инструментами для анализа кода, профиляции и эмуляции, важным классом которых являются динамические бинарные трансляторы (далее — ДБТ)
- В качестве промежуточного представления для ДБТ хорошо использовать LLVM IR
  - Тулчейн LLVM позволяет проводить качественные оптимизации
- Instrew является ДБТ с поднятием кода в LLVM IR, но нет поддержки RISC-V, как хост-архитектуры
- Результатом работы является добавление поддержки архитектуры RISC-V для ДБТ Instrew с открытым исходным кодом
- В связи с наличием процессорно-специфических и низкоуровневых деталей реализации, задача является довольно трудоемкой

# Постановка задачи

**Цель:** Добавить поддержку архитектуры RISC-V для ДБТ Instrew.

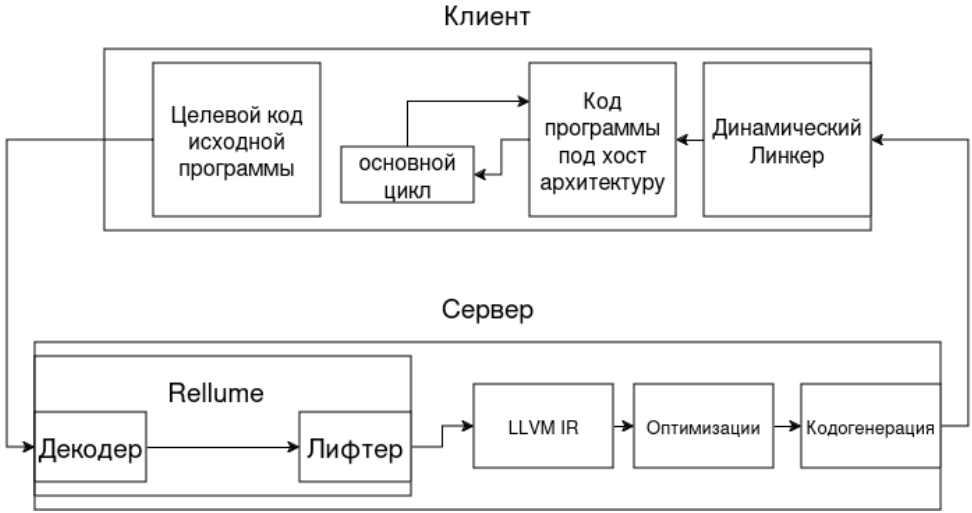
**Задачи:**

1. Выполнить обзор динамических бинарных трансляторов и сравнить их с Instrew
2. Выполнить обзор архитектуры Instrew
3. Реализовать минимальную функциональность для запуска Instrew на RISC-V:
   - Добавить процессорно-специфические патчи в minilibc и другие компоненты
   - Реализовать функции, связанные с RISC-V ABI
4. Провести тестирование

# Обзор

- Динамические бинарные трансляторы:
  - QEMU
  - Box64
  - Rosetta 2
  - Valgrind
- Динамические бинарные трансляторы с поднятием в LLVM IR:
  - Remill + McSema
  - Instrew + Rellume
  - DBILL

# Архитектура Instrew



# Пример (1/4)

Listing 1: objdump -d exit.S

```
Disassembly of section .text
000000000001010c <_start>:
1010c:        4501                    li      a0,0
1010e:        05e00893                li      a7,94
10112:        00000073                ecall
10116:        a001                    j       10116 <_start+0xa>
```

# Пример (2/4)

Listing 2: ./build/server/instrew –dumpir=lift ./test/riscv64/exit

```
define void @S0_1010c(ptr addrspace(1) noalias nocapture align 16
    dereferenceable(400) %0) #0 {
  %2 = getelementptr i8, ptr addrspace(1) %0, i64 0
    ..............
  %66 = getelementptr i8, ptr addrspace(1) %0, i64 512
  %67 = load i64, ptr addrspace(1) %0, align 8
  br label %68
68:                                               ; preds = %1
  store i64 65814, ptr addrspace(1) %0, align 8
  store i64 0, ptr addrspace(1) %13, align 8
  store i64 94, ptr addrspace(1) %20, align 8
  call void @syscall_rv64(ptr addrspace(1) %0)
  %69 = load i64, ptr addrspace(1) %0, align 8
  br label %70
70:                                               ; preds = %68
  %71 = phi i64 [ %69, %68 ]
```

# Пример (3/4)

Listing 3: ./build/server/instrew –dumpir=codegen ./test/riscv64/exit

```
declare void @syscall_rv64(ptr addrspace(1))
; Function Attrs: null_pointer_is_valid
define void @S0_1010c(ptr addrspace(1) noalias nocapture align 16
    dereferenceable(400) %0) #0 {
  br label %2
2:                                                ; preds = %1
  %3 = getelementptr inbounds i8, ptr addrspace(1) %0, i64 144
  %4 = getelementptr inbounds i8, ptr addrspace(1) %0, i64 88
  store i64 65814, ptr addrspace(1) %0, align 16
  store i64 0, ptr addrspace(1) %4, align 8
  store i64 94, ptr addrspace(1) %3, align 16
  call void @syscall_rv64(ptr addrspace(1) %0)
  br label %5
5:                                                ; preds = %2
  ret void
```

# Пример (3/4)

Listing 4: ./build/server/instrew –dumpobj ./test/riscv64/exit

```
0000000000000000 <S0_1010c>:
   0:   ff010113                addi    sp,sp,-16
   4:   00113423                sd      ra,8(sp)
   8:   000105b7                lui     a1,0x10
   c:   1165859b                addiw   a1,a1,278 # 10116 <S0_1010c+0x10116>
  10:   00b53023                sd      a1,0(a0)
  14:   04053c23                sd      zero,88(a0)
  18:   05e00593                li      a1,94
  1c:   08b53823                sd      a1,144(a0)
  20:   00000097                auipc   ra,0x0
  24:   000080e7                jalr    ra # 20 <S0_1010c+0x20>
  28:   00813083                ld      ra,8(sp)
  2c:   01010113                addi    sp,sp,16
  30:   00008067                ret
```

# Реализация (1/3)

Listing 5: Релокация

```
uint64_t syma = (uintptr_t) sym + patch_data->addend;
uint64_t pc = patch_data->patch_addr;
int64_t prel_syma = syma - (int64_t) pc;

case R_RISCV_32_PCREL:

if (!rtld_elf_signed_range(prel_syma, 32, "R_RISCV_32_PCREL"))
    return -EINVAL;
rtld_blend(tgt, 0xffffffffff, prel_syma);
break;
```

# Реализация (2/3)

Listing 6: Заголовок

```
ASM_BLOCK(
    .text;
    .global _start;
    .type _start, %function;
_start:
    .weak __global_pointer$;
    .hidden __global_pointer$;
    .option push;
    .option norelax;
    lla gp, __global_pointer$;
    .option pop;
    mv a0, sp;
    .weak _DYNAMIC;
    .hidden _DYNAMIC;
    lla a1, _DYNAMIC;
    andi sp, sp, -16;
    tail __start_main;  );
```

# Реализация (3/3)

Listing 7: Системный вызов

```
static size_t syscall2(int n, size_t a, size_t b)
{
    register size_t a7 __asm__("a7") = n;
    register size_t a0 __asm__("a0") = a;
    register size_t a1 __asm__("a1") = b;
    __asm_syscall("r"(a7), "0"(a0), "r"(a1))
}
```

# Заключение

В результате работы были выполнены следующие задачи:

1. Выполнен обзор архитектуры Instrew и других ДБТ
2. Реализованы некоторые процессорно-специфические патчи
3. Реализована эмуляция системных вызовов
4. Реализованы архитектурно-специфичные ELF релокации

Протестировать Instrew на простых примерах не удалось, из-за сложности локализации ошибки в реализации релокаций