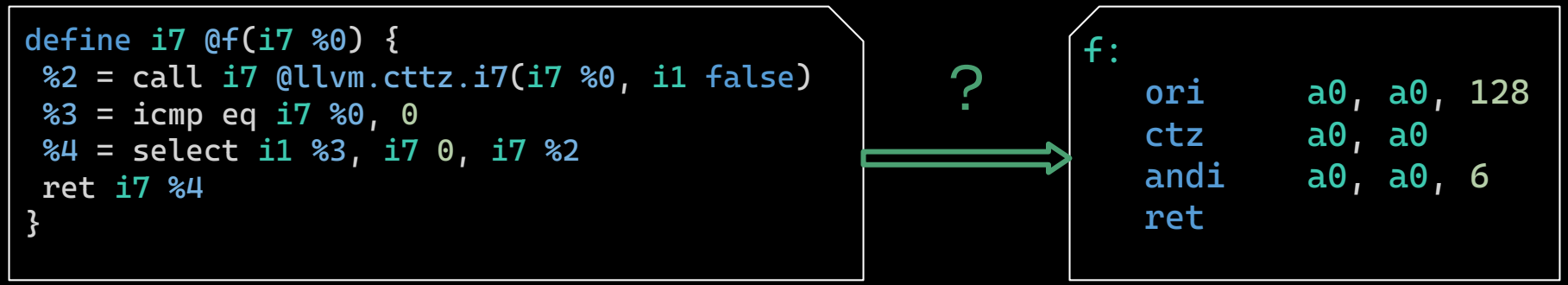
Translation validation for LLVM's RISC-V Backend

Mitch Briles John Regehr



Alive2

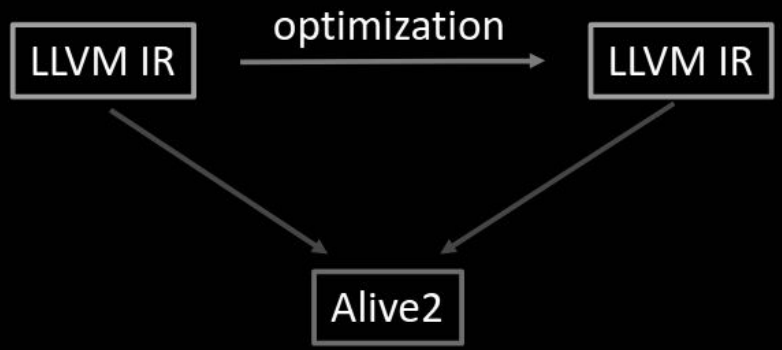
alive-tv formally validates refinements

alive2.llvm.org

```
define i32 @src(i32 %#0) {
#1:
  %r = udiv i32 %#0, 8192
  ret i32 %r
}
⟹
define i32 @tgt(i32 %#0) {
#1:
  %r = lshr i32 %#0, 13
  ret i32 %r
}
Transformation seems to be
correct!
```

Alive2

- Alive2
  - Verifies optimizations using z3
  - Designed to verify middle-end optimizations



ARM-TV

```
define i8 @src(i8 %shamt) {
entry:
  %shl = shl i8 1, %shamt
  ret i8 %shl
}
```

Step 4: Verify refinement with Alive2

```
define i8 @tgt(i8 %0) {
entry:
  %1 = and i8 %0, 31
  %a5_5 = zext nneg i8 %1 to i32
  %a5_6 = shl nuw i32 1, %a5_5
  %a6_2 = trunc i32 %a5_6 to i8
  ret i8 %a6_2
}
```

Step 1: lower with LLVM's AArch64 backend

```
src:
  mov   w8, #1
  lsl   w0, w8, w0
  ret
```

a bunch of IR

Step 2: lift assembly back to LLVM IR

Step 3: optimize lifted IR with middle-end

RISCV-TV

```
define i8 @src(i8 %shamt) {
entry:
  %shl = shl i8 1, %shamt
  ret i8 %shl
}
```

Step 4: Verify refinement with Alive2

```
define i8 @tgt(i8 %0) {
entry:
  %1 = and i8 %0, 63
  %a3_3 = zext nneg i8 %1 to i64
  %a3_4 = shl nuw i64 1, %a3_3
  %a4_2 = trunc i64 %a3_4 to i8
  ret i8 %a4_2
}
```

Step 1: lower with LLVM's RISC-V backend

```
src:
  bset a0, zero, a0
  ret
```

Step 2: lift assembly back to LLVM IR

a bunch of IR

Step 3: optimize lifted IR with middle-end

How to Lift

- ↳ Allocate storage for registers
- ↳ Allocate stack
- ↳ Emulate ABI
  - ↳ Store arguments in registers
- ↳ Translate each instruction to LLVM IR
  - ↳ Load registers used in instruction
  - ↳ Generate semantically equivalent IR
  - ↳ Store results back in registers

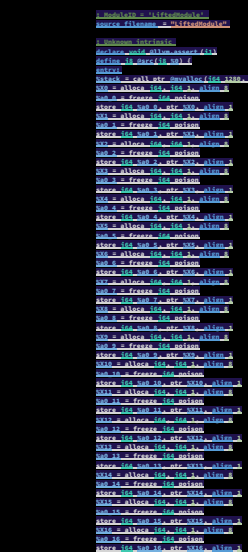
How to Lift

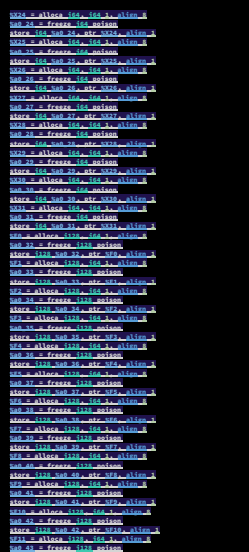
```
case RISCV::ADD: {
  auto a = readFromRegOperand(1, i64ty);
  auto b = readFromRegOperand(2, i64ty);
  res = createAdd(a, b);
   updateOutputReg(res);
  break;
}
```

How to Lift

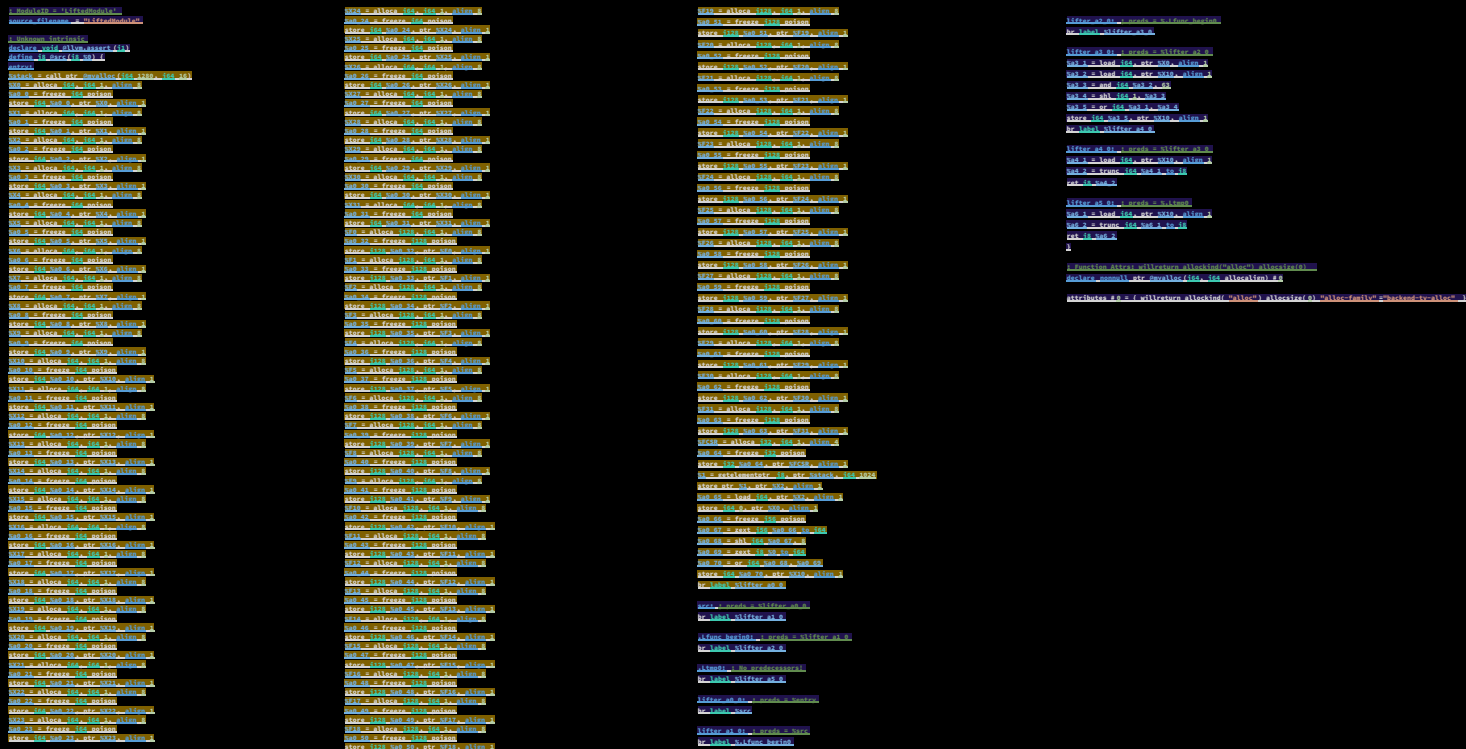
```
case RISCV::ORC_B: {
  auto a = readFromRegOperand(1, i64ty);

  // smear byte to LSB
  auto t1 = createRawLShr(a, getUnsignedIntConst(1, 64));
  auto t1m = createAnd(t1, getUnsignedIntConst(0x7F7F7F7F7F7F7F7F, 64));
  auto s1 = createOr(a, t1m);
  auto t2 = createRawLShr(s1, getUnsignedIntConst(2, 64));
  auto t2m = createAnd(t2, getUnsignedIntConst(0x3F3F3F3F3F3F3F3F, 64));
  auto s2 = createOr(s1, t2m);
  auto t3 = createRawLShr(s2, getUnsignedIntConst(4, 64));
  auto t3m = createAnd(t3, getUnsignedIntConst(0x0F0F0F0F0F0F0F0F, 64));
  auto s3 = createOr(s2, t3m);

  // extract LSB
  auto bits = createAnd(s3, getUnsignedIntConst(0x0101010101010101, 64));

  // scale any 0x01 to 0xFF
  auto res = createMul(bits, getUnsignedIntConst(0xFF, 64));
  updateOutputReg(res);
  break;
}
```

Lifted Code





Lifted Code



Lifted Code

```
src:
  bset a0, zero, a0
  ret
```

=

```
define i8 @tgt(i8 %0) {
  ; register and stack initialization
  %X10 = alloca i64, i64 1, align 8
  %a0_10 = freeze i64 poison
  store i64 %a0_10, ptr %X10, align 1
  < . . . >

lifter_a3_0:
  %a3_1 = load i64, ptr %X0, align 1
  %a3_2 = load i64, ptr %X10, align 1
  %a3_3 = and i64 %a3_2, 63
  %a3_4 = shl i64 1, %a3_3
  %a3_5 = or i64 %a3_1, %a3_4
  store i64 %a3_5, ptr %X10, align 1
  br label %lifter_a4_0

lifter_a4_0:
  %a4_1 = load i64, ptr %X10, align 1
  %a4_2 = trunc i64 %a4_1 to i8
  ret i8 %a4_2
}
```

Where do we get tests?

Where do we get tests?

- LLVM's test suite
  - Contains interesting patterns

Where do we get tests?

- LLVM's test suite
  - Contains interesting patterns
- yarpgen
  - Used to generate random C functions
  - C is compiled to LLVM IR

Where do we get tests?

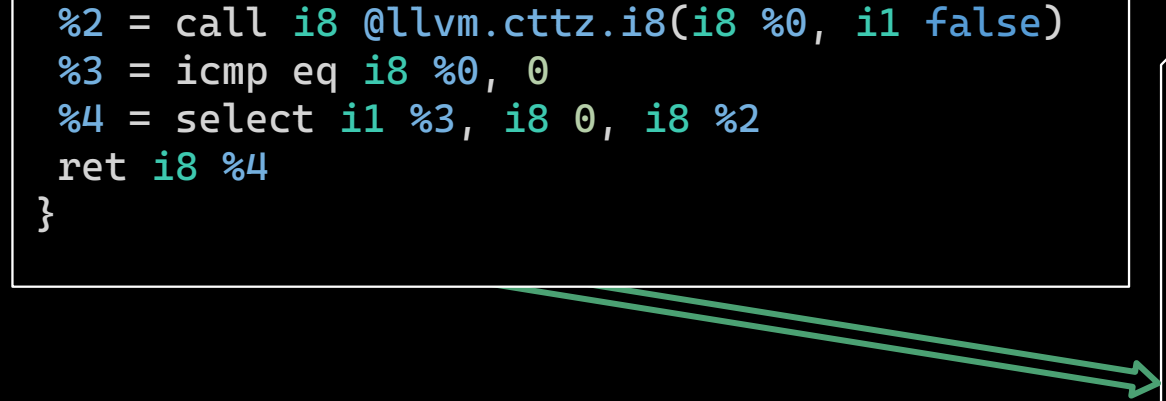
- LLVM's test suite
  - Contains interesting patterns
- yarpgen
  - Used to generate random C functions
  - C is compiled to LLVM IR
- alive-mutate
  - Creates slight variations of existing functions

Where do we get tests?

- LLVM's test suite
  - Contains interesting patterns
- yarpgen
  - Used to generate random C functions
  - C is compiled to LLVM IR
- alive-mutate
  - Creates slight variations of existing functions
- llvm-mutation-based-fuzz-service
  - Has another good mutator

Example

```
define i8 @f(i8 %0) {
 %2 = call i8 @llvm.cttz.i8(i8 %0, i1 false)
 %3 = icmp eq i8 %0, 0
 %4 = select i1 %3, i8 0, i8 %2
 ret i8 %4
}
```



LLVM 21.1.0

```
f:
    ori     a0, a0, 256
    ctz     a0, a0
    andi    a0, a0, 7
    ret
```

Example Bug

```
define i7 @f(i7 %0) {
 %2 = call i7 @llvm.cttz.i7(i7 %0, i1 false)
 %3 = icmp eq i7 %0, 0
 %4 = select i1 %3, i7 0, i7 %2
 ret i7 %4
}
```

[RISCV] Fix incorrect folding of select on ctlz/cttz #155231

Merged lukel97 merged 4 commits into `llvm:main` from `MitchBriles:ctlz-cttz-wrong-fold` on Sep 2

LLVM 21.1.0

```
f:
    andi    a0, a0, 127
    ctz     a0, a0, 128
    andi    a0, a0, 6
    seqz    a1, a1
    addi    a1, a1, -1
    and     a0, a1, a0
    ret
```

Results

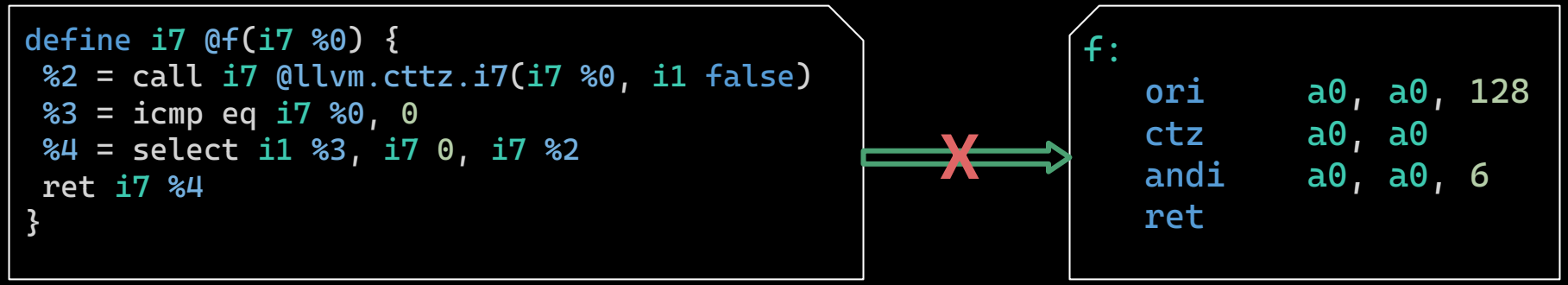
- ARM-TV
  - Found **45** previously unknown miscompilations!
- RISCV-TV
  - Just **1** miscompile specific to RISC-V
  - Detected some unfixed bugs found by ARM-TV

Why so few?

ARM-TV may have caught most of the easier bugs in shared code.

Results

- ↳ Fuzz one, fuzz all!


define i7 @f(i7 %0) {
%2 = call i7 @llvm.cttz.i7(i7 %0, i1 false)
%3 = icmp eq i7 %0, 0
%4 = select i1 %3, i7 0, i7 %2
ret i7 %4
}
f:
ori a0, a0, 128
ctz a0, a0
andi a0, a0, 6
ret

Thank you