

Translation validation for LLVM's RISC-V Backend

Mitch Briles

John Regehr

```
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
```

Alive2



alive-tv formally validates
refinements

alive2.llvm.org

```
define i32 @src(i32 %#0) {  
#1:  
    %r = udiv i32 %#0, 8192  
    ret i32 %r  
}
```

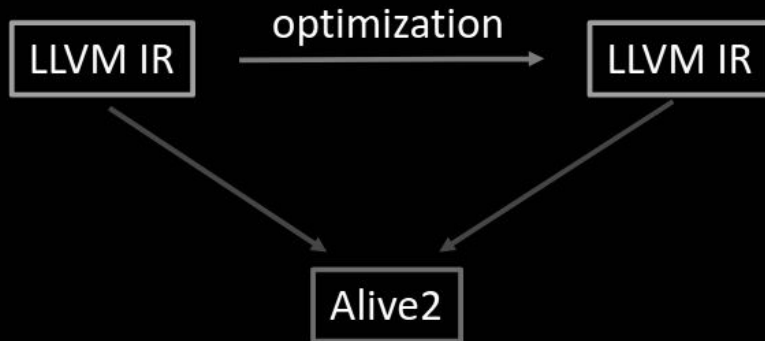
⇒

```
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
```

Step 2: lift assembly
back to LLVM IR

a bunch of 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 1: lower with
LLVM's RISC-V
backend

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

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 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

[illegible]

[1997-1998](#) [1998-1999](#) [1999-2000](#) [2000-2001](#) [2001-2002](#) [2002-2003](#) [2003-2004](#) [2004-2005](#) [2005-2006](#) [2006-2007](#) [2007-2008](#) [2008-2009](#) [2009-2010](#) [2010-2011](#) [2011-2012](#) [2012-2013](#) [2013-2014](#) [2014-2015](#) [2015-2016](#) [2016-2017](#) [2017-2018](#) [2018-2019](#) [2019-2020](#) [2020-2021](#) [2021-2022](#) [2022-2023](#) [2023-2024](#) [2024-2025](#) [2025-2026](#) [2026-2027](#) [2027-2028](#) [2028-2029](#) [2029-2030](#) [2030-2031](#) [2031-2032](#) [2032-2033](#) [2033-2034](#) [2034-2035](#) [2035-2036](#) [2036-2037](#) [2037-2038](#) [2038-2039](#) [2039-2040](#) [2040-2041](#) [2041-2042](#) [2042-2043](#) [2043-2044](#) [2044-2045](#) [2045-2046](#) [2046-2047](#) [2047-2048](#) [2048-2049](#) [2049-2050](#) [2050-2051](#) [2051-2052](#) [2052-2053](#) [2053-2054](#) [2054-2055](#) [2055-2056](#) [2056-2057](#) [2057-2058](#) [2058-2059](#) [2059-2060](#) [2060-2061](#) [2061-2062](#) [2062-2063](#) [2063-2064](#) [2064-2065](#) [2065-2066](#) [2066-2067](#) [2067-2068](#) [2068-2069](#) [2069-2070](#) [2070-2071](#) [2071-2072](#) [2072-2073](#) [2073-2074](#) [2074-2075](#) [2075-2076](#) [2076-2077](#) [2077-2078](#) [2078-2079](#) [2079-2080](#) [2080-2081](#) [2081-2082](#) [2082-2083](#) [2083-2084](#) [2084-2085](#) [2085-2086](#) [2086-2087](#) [2087-2088](#) [2088-2089](#) [2089-2090](#) [2090-2091](#) [2091-2092](#) [2092-2093](#) [2093-2094](#) [2094-2095](#) [2095-2096](#) [2096-2097](#) [2097-2098](#) [2098-2099](#) [2099-2100](#) [2100-2101](#) [2101-2102](#) [2102-2103](#) [2103-2104](#) [2104-2105](#) [2105-2106](#) [2106-2107](#) [2107-2108](#) [2108-2109](#) [2109-2110](#) [2110-2111](#) [2111-2112](#) [2112-2113](#) [2113-2114](#) [2114-2115](#) [2115-2116](#) [2116-2117](#) [2117-2118](#) [2118-2119](#) [2119-2120](#) [2120-2121](#) [2121-2122](#) [2122-2123](#) [2123-2124](#) [2124-2125](#) [2125-2126](#) [2126-2127](#) [2127-2128](#) [2128-2129](#) [2129-2130](#) [2130-2131](#) [2131-2132](#) [2132-2133](#) [2133-2134](#) [2134-2135](#) [2135-2136](#) [2136-2137](#) [2137-2138](#) [2138-2139](#) [2139-2140](#) [2140-2141](#) [2141-2142](#) [2142-2143](#) [2143-2144](#) [2144-2145](#) [2145-2146](#) [2146-2147](#) [2147-2148](#) [2148-2149](#) [2149-2150](#) [2150-2151](#) [2151-2152](#) [2152-2153](#) [2153-2154](#) [2154-2155](#) [2155-2156](#) [2156-2157](#) [2157-2158](#) [2158-2159](#) [2159-2160](#) [2160-2161](#) [2161-2162](#) [2162-2163](#) [2163-2164](#) [2164-2165](#) [2165-2166](#) [2166-2167](#) [2167-2168](#) [2168-2169](#) [2169-2170](#) [2170-2171](#) [2171-2172](#) [2172-2173](#) [2173-2174](#) [2174-2175](#) [2175-2176](#) [2176-2177](#) [2177-2178](#) [2178-2179](#) [2179-2180](#) [2180-2181](#) [2181-2182](#) [2182-2183](#) [2183-2184](#) [2184-2185](#) [2185-2186](#) [2186-2187](#) [2187-2188](#) [2188-2189](#) [2189-2190](#) [2190-2191](#) [2191-2192](#) [2192-2193](#) [2193-2194](#) [2194-2195](#) [2195-2196](#) [2196-2197](#) [2197-2198](#) [2198-2199](#) [2199-2200](#) [2200-2201](#) [2201-2202](#) [2202-2203](#) [2203-2204](#) [2204-2205](#) [2205-2206](#) [2206-2207](#) [2207-2208](#) [2208-2209](#) [2209-2210](#) [2210-2211](#) [2211-2212](#) [2212-2213](#) [2213-2214](#) [2214-2215](#) [2215-2216](#) [2216-2217](#) [2217-2218](#) [2218-2219](#) [2219-2220](#) [2220-2221](#) [2221-2222](#) [2222-2223](#) [2223-2224](#) [2224-2225](#) [2225-2226](#) [2226-2227](#) [2227-2228](#) [2228-2229](#) [2229-2230](#) [2230-2231](#) [2231-2232](#) [2232-2233](#) [2233-2234](#) [2234-2235](#) [2235-2236](#) [2236-2237](#) [2237-2238](#) [2238-2239](#) [2239-2240](#) [2240-2241](#) [2241-2242](#) [2242-2243](#) [2243-2244](#) [2244-2245](#) [2245-2246](#) [2246-2247](#) [2247-2248](#) [2248-2249](#) [2249-2250](#) [2250-2251](#) [2251-2252](#) [2252-2253</](#)

[illegible][illegible]

Lifted Code

[illegible][illegible][illegible][illegible]

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  
}
```

LLVM Bug

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

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

Merged luke97 merged 4 commits into llvm:main from MitchBriles:ctlz-cttz-wrong-fold on Sep 2

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