Anton Johansson
anjo@rev.ng

Alessandro Di Federico
ale@rev.ng

June 15 – KVM Forum 2023

Research paid for by Qualcomm Innovation Center, Inc.

rev.ng

- We are building a decompiler based on QEMU and LLVM
- We do consultancy on compilers, emulators, and dynamic binary translation

QEMU

You either:

- write an interpreter
- write a JIT compiler translating from guest to host, maybe with an IR

You either:

- write an interpreter
- write a JIT compiler translating from guest to host, maybe with an IR

In QEMU you can do both with helper functions and TCG

From helper functions:

```
void vec_add_bytes(uint8_t *dst_vec,
                   uint8_t *src_vec_a,
                   uint8_t *src_vec_b)
{
    for (int i = 0; i < 128; ++i) {
        dst_vec[i] = src_vec_a[i] + src_vec_b[i];
    }
}
```

From helper functions:

```
void vec_add_bytes(uint8_t *dst_vec,
                   uint8_t *src_vec_a,
                   uint8_t *src_vec_b)
{
    for (int i = 0; i < 128; ++i) {
        dst_vec[i] = src_vec_a[i] + src_vec_b[i];
    }
}
```

To TCG code:

```
void vec_add_bytes(intptr_t vo_0,
                   intptr_t vo_1,
                   intptr_t vo_2) {
    tcg_gen_gvec_add(MO_8,
                     vo_0, vo_2, vo_1,
                     128, 128);
}
```

Background (Hexagon & QEMU)

idef-parser

helper2tcg

Demo

Benchmarks (Hexagon)

Summary

Background (Hexagon & QEMU)

idef-parser

helper2tcg

Demo

Benchmarks (Hexagon)

Summary

Goal: write a fast, open-source emulator for Qualcomm® Hexagon™ DSP[1]

[1] Qualcomm and Hexagon are trademarks or registered trademarks of Qualcomm Incorporated.

- VLIW

- VLIW
- > 2000 user mode instructions

- VLIW
- > 2000 user mode instructions

- VLIW
- > 2000 user mode instructions
- Example Hexagon Assembly (part of inner loop of FFT)

```
{ R17:16 = MEMD(R0++M1)
  MEMD(R6++M1) = R25:24
  R20 = CMPY(R20, R8):<<1:rnd:sat
  R11:10 = VADDH(R11:10, R13:12)
}:endloop0
```

- VLIW
- > 2000 user mode instructions
- Example Hexagon Assembly (part of inner loop of FFT)

```
{ R17:16 = MEMD(R0++M1)
  MEMD(R6++M1) = R25:24
  R20 = CMPY(R20, R8):<<1:rnd:sat
  R11:10 = VADDH(R11:10, R13:12)
}:endloop0
```

- KVM 19: QEMU-Hexagon: Automatic Translation of the ISA Manual Pseudocode to Tiny Code Instructions

Helper:

```
int32_t predicated_add(int32_t dst,
                       int32_t pred,
                       int32_t src1,
                       int32_t src2)
{
    if (pred & 1) {
        dst = src1 + src2;
    }
    return dst;
}
```

Helper:

```
int32_t predicated_add(int32_t dst,
                       int32_t pred,
                       int32_t src1,
                       int32_t src2)
{
    if (pred & 1) {
        dst = src1 + src2;
    }
    return dst;
}
```

Tiny Code:

```
// predicate is p0, src is r1,r2, dst is r3
and_i32 loc1,p0,$0x1
brcond_i32 loc1,$0x0,eq,$L1
add_i32 r3,r1,r2
set_label $L1
```

Helper:

```
int32_t predicated_add(int32_t dst,
                       int32_t pred,
                       int32_t src1,
                       int32_t src2)
{
    if (pred & 1) {
        dst = src1 + src2;
    }
    return dst;
}
```

Tiny Code:

Or..

Helper:

```
int32_t predicated_add(int32_t dst,
                       int32_t pred,
                       int32_t src1,
                       int32_t src2)
{
    if (pred & 1) {
        dst = src1 + src2;
    }
    return dst;
}
```

Tiny Code:

```
// predicate is p0, src is r1,r2, dst is r3
and_i32 loc1,p0,$0x1
add_i32 loc2,r1,r2
movcond_i32 r3,loc1,$0x0,r3,loc2,eq
```

Helper:

```
int32_t predicated_add(int32_t dst,
                       int32_t pred,
                       int32_t src1,
                       int32_t src2)
{
    if (pred & 1) {
        dst = src1 + src2;
    }
    return dst;
}
```

Tiny Code Generator (TCG):

Helper:

```
int32_t predicated_add(int32_t dst,
                       int32_t pred,
                       int32_t src1,
                       int32_t src2)
{
    if (pred & 1) {
        dst = src1 + src2;
    }
    return dst;
}
```

Tiny Code Generator (TCG):

```
void predicated_add(TCGv_i32 dst,
                    TCGv_i32 pred
                    TCGv_i32 src1,
                    TCGv_i32 src2)
{
    TCGv_i32 and = tcg_temp_new_i32();
    tcg_gen_andi_i32(and, pred, 1);

    TCGv_i32 add = tcg_temp_new_i32();
    tcg_gen_add_i32(add_1, src1, src2);

    tcg_gen_movcond_i32(TCG_COND_EQ, dst,
                pred, tcg_constant_i32(0),
                dst, add);
}
```

```
if (p0) r3 = add(r1,r2)
```

```
if (p0) r3 = add(r1,r2)
```

```
if (p0) r3 = add(r1,r2)
```

Helper:

```
mov     0x8(%rbp),%esi
mov     %esi,0x128(%rbp)
mov     $0x3,%r9d
mov     %eax,%r8d
mov     0xc(%rbp),%ecx
mov     0x100(%rbp),%edx
mov     %rbp,%rdi
call    *0xb4(%rip)
mov     %eax,0x128(%rbp)
mov     %eax,0x8(%rbp)
```

```
if (p0) r3 = add(r1,r2)
```

Helper:

```
mov     0x8(%rbp),%esi
mov     %esi,0x128(%rbp)
mov     $0x3,%r9d
mov     %eax,%r8d
mov     0xc(%rbp),%ecx
mov     0x100(%rbp),%edx
mov     %rbp,%rdi
call    *0xb4(%rip)
mov     %eax,0x128(%rbp)
mov     %eax,0x8(%rbp)
```

```
if (p0) r3 = add(r1,r2)
```

Helper:

```
mov     0x8(%rbp),%esi
mov     %esi,0x128(%rbp)
mov     $0x3,%r9d
mov     %eax,%r8d
mov     0xc(%rbp),%ecx
mov     0x100(%rbp),%edx
mov     %rbp,%rdi
call    *0xb4(%rip)
    <predicated_add>:
        add     %r8d,%ecx
        mov     %esi,%eax
        and     $0x1,%edx
        cmovne  %ecx,%eax
        ret
mov     %eax,0x128(%rbp)
mov     %eax,0x8(%rbp)
```

```
if (p0) r3 = add(r1,r2)
```

Helper:

```
mov     0x8(%rbp),%esi
mov     %esi,0x128(%rbp)
mov     $0x3,%r9d
mov     %eax,%r8d
mov     0xc(%rbp),%ecx
mov     0x100(%rbp),%edx
mov     %rbp,%rdi
call    *0xb4(%rip)
    <predicated_add>:
        add     %r8d,%ecx
        mov     %esi,%eax
        and     $0x1,%edx
        cmovne  %ecx,%eax
        ret
mov     %eax,0x128(%rbp)
mov     %eax,0x8(%rbp)
```

```
if (p0) r3 = add(r1,r2)
```

Helper:

```
mov     0x8(%rbp),%esi
mov     %esi,0x128(%rbp)
mov     $0x3,%r9d
mov     %eax,%r8d
mov     0xc(%rbp),%ecx
mov     0x100(%rbp),%edx
mov     %rbp,%rdi
call    *0xb4(%rip)
    <predicated_add>:
        add     %r8d,%ecx
        mov     %esi,%eax
        and     $0x1,%edx
        cmovne  %ecx,%eax
        ret
mov     %eax,0x128(%rbp)
mov     %eax,0x8(%rbp)
```

TCG:

```
mov     0x8(%rbp),%r15d
mov     0xc(%rbp),%ebx
mov     0x100(%rbp),%r11d
and     $0x1,%r11d
add     %r15d,%ebx
cmp     %r13d,%r11d
cmovne  %ebx,%r10d
mov     %r10d,0x128(%rbp)
mov     %r10d,0x8(%rbp)
```

```
if (p0) r3 = add(r1,r2)
```

Helper:

```
mov     0x8(%rbp),%esi
mov     %esi,0x128(%rbp)
mov     $0x3,%r9d
mov     %eax,%r8d
mov     0xc(%rbp),%ecx
mov     0x100(%rbp),%edx
mov     %rbp,%rdi
call    *0xb4(%rip)
    <predicated_add>:
        add     %r8d,%ecx
        mov     %esi,%eax
        and     $0x1,%edx
        cmovne  %ecx,%eax
        ret
mov     %eax,0x128(%rbp)
mov     %eax,0x8(%rbp)
```

TCG:

```
mov     0x8(%rbp),%r15d
mov     0xc(%rbp),%ebx
mov     0x100(%rbp),%r11d
and     $0x1,%r11d
add     %r15d,%ebx
cmp     %r13d,%r11d
cmovne  %ebx,%r10d
mov     %r10d,0x128(%rbp)
mov     %r10d,0x8(%rbp)
```

```
if (p0) r3 = add(r1,r2)
```

Helper:

```
mov     0x8(%rbp),%esi
mov     %esi,0x128(%rbp)
mov     $0x3,%r9d
mov     %eax,%r8d
mov     0xc(%rbp),%ecx
mov     0x100(%rbp),%edx
mov     %rbp,%rdi
call    *0xb4(%rip)
    <predicated_add>:
        add     %r8d,%ecx
        mov     %esi,%eax
        and     $0x1,%edx
        cmovne  %ecx,%eax
        ret
mov     %eax,0x128(%rbp)
mov     %eax,0x8(%rbp)
```

TCG:

```
mov     0x8(%rbp),%r15d
mov     0xc(%rbp),%ebx
mov     0x100(%rbp),%r11d
and     $0x1,%r11d
add     %r15d,%ebx
cmp     %r13d,%r11d
cmovne  %ebx,%r10d
mov     %r10d,0x128(%rbp)
mov     %r10d,0x8(%rbp)
```

```
r1 = #1
r2 = #2
p0 = #0xff
if (p0) r3 = add(r1,r2)
```

```
r1 = #1
r2 = #2
p0 = #0xff
if (p0) r3 = add(r1,r2)
```

Helper:

```
mov     0x8(%rbp),%esi
mov     %esi,0x128(%rbp)
mov     $0x3,%r9d
mov     %eax,%r8d
mov     0xc(%rbp),%ecx
mov     0x100(%rbp),%edx
mov     %rbp,%rdi
call    *0xb4(%rip)
    <predicated_add>:
        add     %r8d,%ecx
        mov     %esi,%eax
        and     $0x1,%edx
        cmovne  %ecx,%eax
        ret
mov     %eax,0x128(%rbp)
mov     %eax,0x8(%rbp)
```

```
r1 = #1
r2 = #2
p0 = #0xff
if (p0) r3 = add(r1,r2)
```

Helper:

```
mov     0x8(%rbp),%esi
mov     %esi,0x128(%rbp)
mov     $0x3,%r9d
mov     %eax,%r8d
mov     0xc(%rbp),%ecx
mov     0x100(%rbp),%edx
mov     %rbp,%rdi
call    *0xb4(%rip)
    <predicated_add>:
        add     %r8d,%ecx
        mov     %esi,%eax
        and     $0x1,%edx
        cmovne  %ecx,%eax
        ret
mov     %eax,0x128(%rbp)
mov     %eax,0x8(%rbp)
```

TCG:

```
movl    $0x3,0x128(%rbp)
movl    $0x3,0x8(%rbp)
```

Helper:

TCG:

Helper:

- ✓ Easy to implement
- × Slow (calls, no optimizations)

TCG:

Helper:

- ✓ Easy to implement
- ✗ Slow (calls, no optimizations)

TCG:

- ✓ Fast
- ✗ Tedious, error prone
- ✗ Barrier to entry

Background (Hexagon & QEMU)

idef-parser

helper2tcg

Demo

Benchmarks (Hexagon)

Summary

Pseudo C

```
A2_paddt(in PuV, RdV,
         in RsV, in RtV)
{
    if (PuV & 1) {
        RdV=RsV+RtV;
    } else {
        CANCEL;
    }
}
```

Pseudo C

```
A2_paddt(in PuV, RdV,
         in RsV, in RtV)
{
    if (PuV & 1) {
        RdV=RsV+RtV;
    } else {
        CANCEL;
    }
}
```

Pseudo C

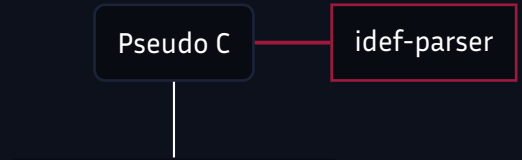
```
A2_paddt(in PuV, RdV,
         in RsV, in RtV)
{
    if (PuV & 1) {
        RdV=RsV+RtV;
    } else {
        CANCEL;
    }
}
```

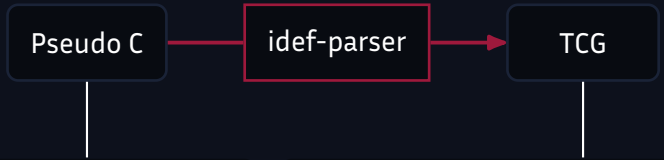
Pseudo C

```
A2_paddt(in PuV, RdV,
         in RsV, in RtV)
{
    if (PuV & 1) {
        RdV=RsV+RtV;
    } else {
        CANCEL;
    }
}
```

Pseudo C

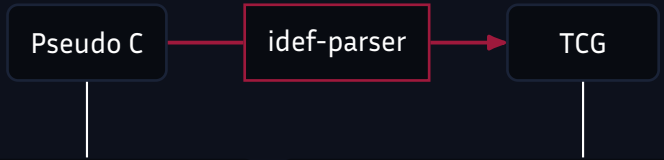
```
A2_paddt(in PuV, RdV,
         in RsV, in RtV)
{
    if (PuV & 1) {
        RdV=RsV+RtV;
    } else {
        CANCEL;
    }
}
```


Pseudo C
idef-parser
A2_paddt(in PuV, RdV,
in RsV, in RtV)
{
if (PuV & 1) {
RdV=RsV+RtV;
} else {
CANCEL;
}
}



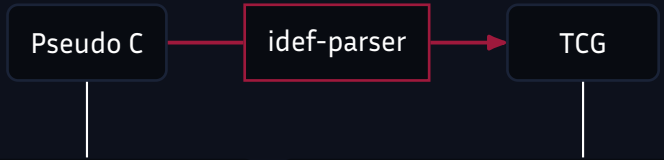

```
A2_paddt(in PuV, RdV,
         in RsV, in RtV)
{
    if (PuV & 1) {
        RdV=RsV+RtV;
    } else {
        CANCEL;
    }
}
```

```
void A2_paddt(TCGv PuV,
              TCGv_i32 RdV,
              TCGv_i32 RsV,
              TCGv_i32 RtV)
{
    TCGv_i32 tmp_0 = tcg_temp_new_i32();
    tcg_gen_extract_i32(tmp_0, PuV, ((int32_t) 0x0) * 1, 1);
    TCGLabel *if_label_0 = gen_new_label();
    tcg_gen_brcondi_i32(TCG_COND_EQ, tmp_0, 0, if_label_0);
    TCGv_i32 tmp_1 = tcg_temp_new_i32();
    tcg_gen_add_i32(tmp_1, RsV, RtV);
    tcg_gen_mov_i32(RdV, tmp_1);
    TCGLabel *if_label_1 = gen_new_label();
    tcg_gen_br(if_label_1);
    gen_set_label(if_label_0);
    gen_set_label(if_label_1);
}
```



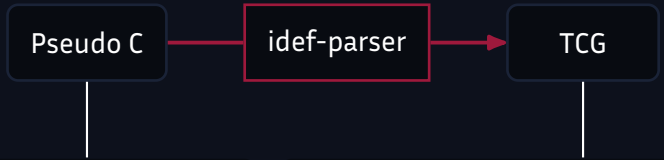

```
A2_paddt(in PuV, RdV,
         in RsV, in RtV)
{
    if (PuV & 1) {
        RdV=RsV+RtV;
    } else {
        CANCEL;
    }
}
```

```
void A2_paddt(TCGv PuV,
              TCGv_i32 RdV,
              TCGv_i32 RsV,
              TCGv_i32 RtV)
{
    TCGv_i32 tmp_0 = tcg_temp_new_i32();
    tcg_gen_extract_i32(tmp_0, PuV, ((int32_t) 0x0) * 1, 1);
    TCGLabel *if_label_0 = gen_new_label();
    tcg_gen_brcondi_i32(TCG_COND_EQ, tmp_0, 0, if_label_0);
    TCGv_i32 tmp_1 = tcg_temp_new_i32();
    tcg_gen_add_i32(tmp_1, RsV, RtV);
    tcg_gen_mov_i32(RdV, tmp_1);
    TCGLabel *if_label_1 = gen_new_label();
    tcg_gen_br(if_label_1);
    gen_set_label(if_label_0);
    gen_set_label(if_label_1);
}
```



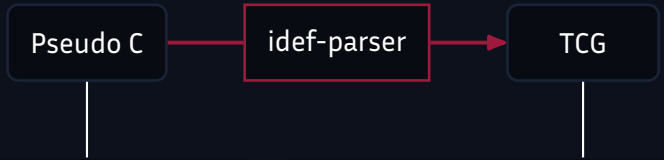

```
A2_paddt(in PuV, RdV,
         in RsV, in RtV)
{
    if (PuV & 1) {
        RdV=RsV+RtV;
    } else {
        CANCEL;
    }
}
```

```
void A2_paddt(TCGv PuV,
              TCGv_i32 RdV,
              TCGv_i32 RsV,
              TCGv_i32 RtV)
{
    TCGv_i32 tmp_0 = tcg_temp_new_i32();
    tcg_gen_extract_i32(tmp_0, PuV, ((int32_t) 0x0) * 1, 1);
    TCGLabel *if_label_0 = gen_new_label();
    tcg_gen_brcondi_i32(TCG_COND_EQ, tmp_0, 0, if_label_0);
    TCGv_i32 tmp_1 = tcg_temp_new_i32();
    tcg_gen_add_i32(tmp_1, RsV, RtV);
    tcg_gen_mov_i32(RdV, tmp_1);
    TCGLabel *if_label_1 = gen_new_label();
    tcg_gen_br(if_label_1);
    gen_set_label(if_label_0);
    gen_set_label(if_label_1);
}
```



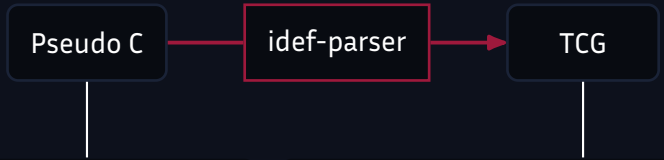

```
A2_paddt(in PuV, RdV,
         in RsV, in RtV)
{
    if (PuV & 1) {
        RdV=RsV+RtV;
    } else {
        CANCEL;
    }
}
```

```
void A2_paddt(TCGv PuV,
              TCGv_i32 RdV,
              TCGv_i32 RsV,
              TCGv_i32 RtV)
{
    TCGv_i32 tmp_0 = tcg_temp_new_i32();
    tcg_gen_extract_i32(tmp_0, PuV, ((int32_t) 0x0) * 1, 1);
    TCGLabel *if_label_0 = gen_new_label();
    tcg_gen_brcondi_i32(TCG_COND_EQ, tmp_0, 0, if_label_0);
    TCGv_i32 tmp_1 = tcg_temp_new_i32();
    tcg_gen_add_i32(tmp_1, RsV, RtV);
    tcg_gen_mov_i32(RdV, tmp_1);
    TCGLabel *if_label_1 = gen_new_label();
    tcg_gen_br(if_label_1);
    gen_set_label(if_label_0);
    gen_set_label(if_label_1);
}
```



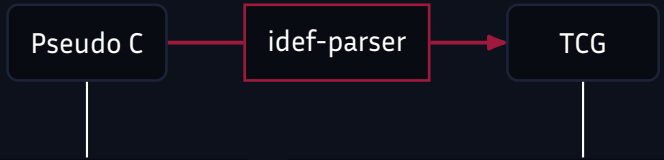

```
A2_paddt(in PuV, RdV,
         in RsV, in RtV)
{
    if (PuV & 1) {
        RdV=RsV+RtV;
    } else {
        CANCEL;
    }
}
```

```
void A2_paddt(TCGv PuV,
              TCGv_i32 RdV,
              TCGv_i32 RsV,
              TCGv_i32 RtV)
{
    TCGv_i32 tmp_0 = tcg_temp_new_i32();
    tcg_gen_extract_i32(tmp_0, PuV, ((int32_t) 0x0) * 1, 1);
    TCGLabel *if_label_0 = gen_new_label();
    tcg_gen_brcondi_i32(TCG_COND_EQ, tmp_0, 0, if_label_0);
    TCGv_i32 tmp_1 = tcg_temp_new_i32();
    tcg_gen_add_i32(tmp_1, RsV, RtV);
    tcg_gen_mov_i32(RdV, tmp_1);
    TCGLabel *if_label_1 = gen_new_label();
    tcg_gen_br(if_label_1);
    gen_set_label(if_label_0);
    gen_set_label(if_label_1);
}
```




```
A2_paddt(in PuV, RdV,
         in RsV, in RtV)
{
    if (PuV & 1) {
        RdV=RsV+RtV;
    } else {
        CANCEL;
    }
}
```

```
void A2_paddt(TCGv PuV,
              TCGv_i32 RdV,
              TCGv_i32 RsV,
              TCGv_i32 RtV)
{
    TCGv_i32 tmp_0 = tcg_temp_new_i32();
    tcg_gen_extract_i32(tmp_0, PuV, ((int32_t) 0x0) * 1, 1);
    TCGLabel *if_label_0 = gen_new_label();
    tcg_gen_brcondi_i32(TCG_COND_EQ, tmp_0, 0, if_label_0);
    TCGv_i32 tmp_1 = tcg_temp_new_i32();
    tcg_gen_add_i32(tmp_1, RsV, RtV);
    tcg_gen_mov_i32(RdV, tmp_1);
    TCGLabel *if_label_1 = gen_new_label();
    tcg_gen_br(if_label_1);
    gen_set_label(if_label_0);
    gen_set_label(if_label_1);
}
```




```
A2_paddt(in PuV, RdV,
         in RsV, in RtV)
{
    if (PuV & 1) {
        RdV=RsV+RtV;
    } else {
        CANCEL;
    }
}
```

```
void A2_paddt(TCGv PuV,
              TCGv_i32 RdV,
              TCGv_i32 RsV,
              TCGv_i32 RtV)
{
    TCGv_i32 tmp_0 = tcg_temp_new_i32();
    tcg_gen_extract_i32(tmp_0, PuV, ((int32_t) 0x0) * 1, 1);
    TCGLabel *if_label_0 = gen_new_label();
    tcg_gen_brcondi_i32(TCG_COND_EQ, tmp_0, 0, if_label_0);
    TCGv_i32 tmp_1 = tcg_temp_new_i32();
    tcg_gen_add_i32(tmp_1, RsV, RtV);
    tcg_gen_mov_i32(RdV, tmp_1);
    TCGLabel *if_label_1 = gen_new_label();
    tcg_gen_br(if_label_1);
    gen_set_label(if_label_0);
    gen_set_label(if_label_1);
}
```

- Written in flex and bison

- Written in flex and bison
- Currently upstreamed and maintained

- Written in flex and bison
- Currently upstreamed and maintained
- Results:

- Written in flex and bison
- Currently upstreamed and maintained
- Results:
  - ✓ Able to generate TCG for 1300 instructions

- Written in flex and bison
- Currently upstreamed and maintained
- Results:
  - ✓ Able to generate TCG for 1300 instructions
  - × Heavily tailored to Hexagon

- Written in flex and bison
- Currently upstreamed and maintained
- Results:
  - ✓ Able to generate TCG for 1300 instructions
  - × Heavily tailored to Hexagon
  - × No AST or IR ⇒ no optimization

- Written in flex and bison
- Currently upstreamed and maintained
- Results:
  - ✓ Able to generate TCG for 1300 instructions
  - × Heavily tailored to Hexagon
  - × No AST or IR ⇒ no optimization
  - × Need to write a poor man's C parser

Background (Hexagon & QEMU)

idef-parser

helper2tcg

Demo

Benchmarks (Hexagon)

Summary

Instead of:

Instead of:

Why not:

C

Instead of:

Why not:

Instead of:

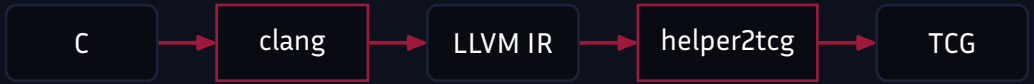
Why not:

Instead of:

Why not:

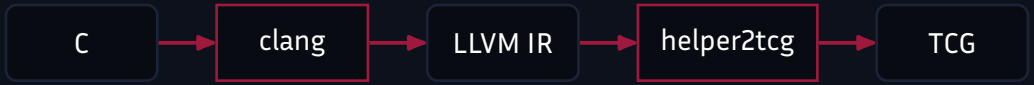
Instead of:

Why not:



Instead of:

Why not:



- ✓ No need for a poor man's C frontend
- ✓ Translating IR $\rightarrow$ IR simpler
- ✓ LLVM optimizations
- ✓ Architecture independent

```c
int32_t predicated_add(int32_t dst,
                       int32_t pred,
                       int32_t src1,
                       int32_t src2)
{
    if (pred & 1) {
        dst = src1 + src2;
    }
    return dst;
}
```

```
define i32 @predicated_add(i32 %dst, i32 %pred,
                           i32 %src1, i32 %src2) {
  %4 = alloca i32
  store i32 %dst, i32* %4

  %5 = and i32 %pred, 1
  %6 = icmp ne i32 %5, 0
  br i1 %6, label %if-label, label %exit-label

if-label:
  %8 = add i32 %src1, %src2
  store i32 %8, i32* %4
  br %exit-label

exit-label:
  %10 = load i32, i32* %4
  ret i32 %10
}
```

```c
int32_t predicated_add(int32_t dst,
                       int32_t pred,
                       int32_t src1,
                       int32_t src2)
{
    if (pred & 1) {
        dst = src1 + src2;
    }
    return dst;
}
```

```
define i32 @predicated_add(i32 %dst, i32 %pred,
                           i32 %src1, i32 %src2) {
  %4 = alloca i32
  store i32 %dst, i32* %4

  %5 = and i32 %pred, 1
  %6 = icmp ne i32 %5, 0
  br i1 %6, label %if-label, label %exit-label

if-label:
  %8 = add i32 %src1, %src2
  store i32 %8, i32* %4
  br %exit-label

exit-label:
  %10 = load i32, i32* %4
  ret i32 %10
}
```

```c
int32_t predicated_add(int32_t dst,
                       int32_t pred,
                       int32_t src1,
                       int32_t src2)
{
    if (pred & 1) {
        dst = src1 + src2;
    }
    return dst;
}
```

```
define i32 @predicated_add(i32 %dst, i32 %pred,
                           i32 %src1, i32 %src2) {
  %4 = alloca i32
  store i32 %dst, i32* %4

  %5 = and i32 %pred, 1
  %6 = icmp ne i32 %5, 0
  br i1 %6, label %if-label, label %exit-label

if-label:
  %8 = add i32 %src1, %src2
  store i32 %8, i32* %4
  br %exit-label

exit-label:
  %10 = load i32, i32* %4
  ret i32 %10
}
```

```c
int32_t predicated_add(int32_t dst,
                       int32_t pred,
                       int32_t src1,
                       int32_t src2)
{
    if (pred & 1) {
        dst = src1 + src2;
    }
    return dst;
}
```



```
define i32 @predicated_add(i32 %dst, i32 %pred,
                           i32 %src1, i32 %src2) {
  %4 = alloca i32
  store i32 %dst, i32* %4

  %5 = and i32 %pred, 1
  %6 = icmp ne i32 %5, 0
  br i1 %6, label %if-label, label %exit-label

if-label:
  %8 = add i32 %src1, %src2
  store i32 %8, i32* %4
  br %exit-label

exit-label:
  %10 = load i32, i32* %4
  ret i32 %10
}
```

```c
int32_t predicated_add(int32_t dst,
                       int32_t pred,
                       int32_t src1,
                       int32_t src2)
{
    if (pred & 1) {
        dst = src1 + src2;
    }
    return dst;
}
```



```
define i32 @predicated_add(i32 %dst, i32 %pred,
                           i32 %src1, i32 %src2) {
  %4 = alloca i32
  store i32 %dst, i32* %4

  %5 = and i32 %pred, 1
  %6 = icmp ne i32 %5, 0
  br i1 %6, label %if-label, label %exit-label

if-label:
  %8 = add i32 %src1, %src2
  store i32 %8, i32* %4
  br %exit-label

exit-label:
  %10 = load i32, i32* %4
  ret i32 %10
}
```

```c
int32_t predicated_add(int32_t dst,
                       int32_t pred,
                       int32_t src1,
                       int32_t src2)
{
    if (pred & 1) {
        dst = src1 + src2;
    }
    return dst;
}
```

```
define i32 @predicated_add(i32 %dst, i32 %pred,
                           i32 %src1, i32 %src2) {
  %4 = alloca i32
  store i32 %dst, i32* %4

  %5 = and i32 %pred, 1
  %6 = icmp ne i32 %5, 0
  br i1 %6, label %if-label, label %exit-label

if-label:
  %8 = add i32 %src1, %src2
  store i32 %8, i32* %4
  br label %exit-label

exit-label:
  %10 = load i32, i32* %4
  ret i32 %10
}
```

```
define i32 @predicated_add(i32 %dst, i32 %pred,
                           i32 %src1, i32 %src2) {
  %4 = alloca i32
  store i32 %dst, i32* %4

  %5 = and i32 %pred, 1
  %6 = icmp ne i32 %5, 0
  br i1 %6, label %if-label, label %exit-label

if-label:
  %8 = add i32 %src1, %src2
  store i32 %8, i32* %4
  br %exit-label

exit-label:
  %10 = load i32, i32* %4
  ret i32 %10
}
```

```
mov_i32 loc1, dst

and_i32 loc2, pred, $0x1
setcond_i32 loc3, pred, $0x0, ne
brcond_i32 loc3, $0x0, eq, $L0
br $L1

set_label $L0
add_i32 loc4, src1, src2
mov_i32 loc1, loc4
br $L1

set_label $L1
mov_i32 dst, loc1
```

```
define i32 @predicated_add(i32 %dst, i32 %pred,
                           i32 %src1, i32 %src2) {
  %4 = alloca i32
  store i32 %dst, i32* %4

  %5 = and i32 %pred, 1
  %6 = icmp ne i32 %5, 0
  br i1 %6, label %if-label, label %exit-label

if-label:
  %8 = add i32 %src1, %src2
  store i32 %8, i32* %4
  br %exit-label

exit-label:
  %10 = load i32, i32* %4
  ret i32 %10
}
```

```
mov_i32 loc1, dst

and_i32 loc2, pred, $0x1
setcond_i32 loc3, pred, $0x0, ne
brcond_i32 loc3, $0x0, eq, $L0
br $L1

set_label $L0
add_i32 loc4, src1, src2
mov_i32 loc1, loc4
br $L1

set_label $L1
mov_i32 dst, loc1
```

```
define i32 @predicated_add(i32 %dst, i32 %pred,
                           i32 %src1, i32 %src2) {
  %4 = alloca i32
  store i32 %dst, i32* %4

  %5 = and i32 %pred, 1
  %6 = icmp ne i32 %5, 0
  br i1 %6, label %if-label, label %exit-label

if-label:
  %8 = add i32 %src1, %src2
  store i32 %8, i32* %4
  br %exit-label

exit-label:
  %10 = load i32, i32* %4
  ret i32 %10
}
```

```
mov_i32 loc1, dst

and_i32 loc2, pred, $0x1
setcond_i32 loc3, pred, $0x0, ne
brcond_i32 loc3, $0x0, eq, $L0
br $L1

set_label $L0
add_i32 loc4, src1, src2
mov_i32 loc1, loc4
br $L1

set_label $L1
mov_i32 dst, loc1
```

```
define i32 @predicated_add(i32 %dst, i32 %pred,
                           i32 %src1, i32 %src2) {
  %4 = alloca i32
  store i32 %dst, i32* %4

  %5 = and i32 %pred, 1
  %6 = icmp ne i32 %5, 0
  br i1 %6, label %if-label, label %exit-label

if-label:
  %8 = add i32 %src1, %src2
  store i32 %8, i32* %4
  br %exit-label

exit-label:
  %10 = load i32, i32* %4
  ret i32 %10
}
```

```
mov_i32 loc1, dst

and_i32 loc2, pred, $0x1
setcond_i32 loc3, pred, $0x0, ne
brcond_i32 loc3, $0x0, eq, $L0
br $L1

set_label $L0
add_i32 loc4, src1, src2
mov_i32 loc1, loc4
br $L1

set_label $L1
mov_i32 dst, loc1
```

```
define i32 @predicated_add(i32 %dst, i32 %pred,
                           i32 %src1, i32 %src2) {
  %4 = alloca i32
  store i32 %dst, i32* %4

  %5 = and i32 %pred, 1
  %6 = icmp ne i32 %5, 0
  br i1 %6, label %if-label, label %exit-label

if-label:
  %8 = add i32 %src1, %src2
  store i32 %8, i32* %4
  br %exit-label

exit-label:
  %10 = load i32, i32* %4
  ret i32 %10
}
```

```
mov_i32 loc1, dst

and_i32 loc2, pred, $0x1
setcond_i32 loc3, pred, $0x0, ne
brcond_i32 loc3, $0x0, eq, $L0
br $L1

set_label $L0
add_i32 loc4, src1, src2
mov_i32 loc1, loc4
br $L1

set_label $L1
mov_i32 dst, loc1
```

```
define i32 @predicated_add(i32 %dst, i32 %pred,
                           i32 %src1, i32 %src2) {
  %4 = alloca i32
  store i32 %dst, i32* %4

  %5 = and i32 %pred, 1
  %6 = icmp ne i32 %5, 0
  br i1 %6, label %if-label, label %exit-label

if-label:
  %8 = add i32 %src1, %src2
  store i32 %8, i32* %4
  br %exit-label

exit-label:
  %10 = load i32, i32* %4
  ret i32 %10
}
```

```
mov_i32 loc1, dst

and_i32 loc2, pred, $0x1
setcond_i32 loc3, pred, $0x0, ne
brcond_i32 loc3, $0x0, eq, $L0
br $L1

set_label $L0
add_i32 loc4, src1, src2
mov_i32 loc1, loc4
br $L1

set_label $L1
mov_i32 dst, loc1
```

```
define i32 @predicated_add(i32 %dst, i32 %pred,
                           i32 %src1, i32 %src2) {
  %4 = alloca i32
  store i32 %dst, i32* %4

  %5 = and i32 %pred, 1
  %6 = icmp ne i32 %5, 0
  br i1 %6, label %if-label, label %exit-label

if-label:
  %8 = add i32 %src1, %src2
  store i32 %8, i32* %4
  br %exit-label

exit-label:
  %10 = load i32, i32* %4
  ret i32 %10
}
```

```
mov_i32 loc1, dst

and_i32 loc2, pred, $0x1
setcond_i32 loc3, pred, $0x0, ne
brcond_i32 loc3, $0x0, eq, $L0
br $L1

set_label $L0
add_i32 loc4, src1, src2
mov_i32 loc1, loc4
br $L1

set_label $L1
mov_i32 dst, loc1
```

```
define i32 @predicated_add(i32 %dst, i32 %pred,
                           i32 %src1, i32 %src2) {
  %4 = alloca i32
  store i32 %dst, i32* %4

  %5 = and i32 %pred, 1
  %6 = icmp ne i32 %5, 0
  br i1 %6, label %if-label, label %exit-label

if-label:
  %8 = add i32 %src1, %src2
  store i32 %8, i32* %4
  br %exit-label

exit-label:
  %10 = load i32, i32* %4
  ret i32 %10
}
```

```
mov_i32 loc1, dst

and_i32 loc2, pred, $0x1
setcond_i32 loc3, pred, $0x0, ne
brcond_i32 loc3, $0x0, eq, $L0
br $L1

set_label $L0
add_i32 loc4, src1, src2
mov_i32 loc1, loc4
br $L1

set_label $L1
mov_i32 dst, loc1
```

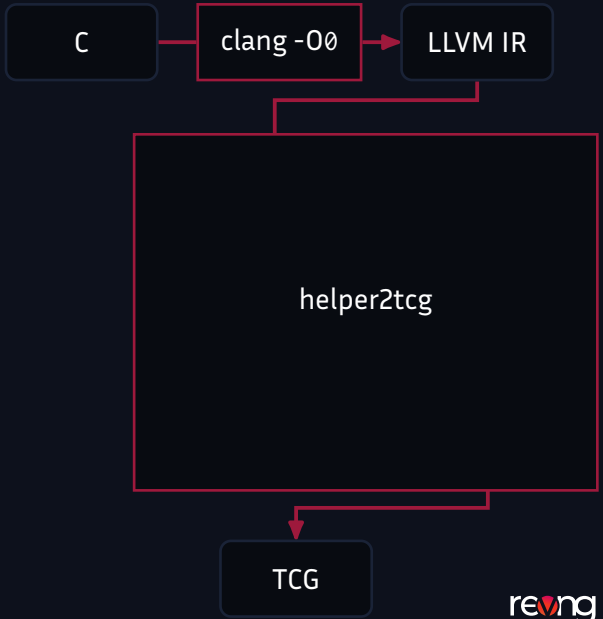
```
define i32 @predicated_add(i32 %dst, i32 %pred,
                           i32 %src1, i32 %src2) {
  %4 = alloca i32
  store i32 %dst, i32* %4

  %5 = and i32 %pred, 1
  %6 = icmp ne i32 %5, 0
  br i1 %6, label %if-label, label %exit-label

if-label:
  %8 = add i32 %src1, %src2
  store i32 %8, i32* %4
  br %exit-label

exit-label:
  %10 = load i32, i32* %4
  ret i32 %10
}
```

```
mov_i32 loc1, dst

and_i32 loc2, pred, $0x1
setcond_i32 loc3, pred, $0x0, ne
brcond_i32 loc3, $0x0, eq, $L0
br $L1

set_label $L0
add_i32 loc4, src1, src2
mov_i32 loc1, loc4
br $L1

set_label $L1
mov_i32 dst, loc1
```

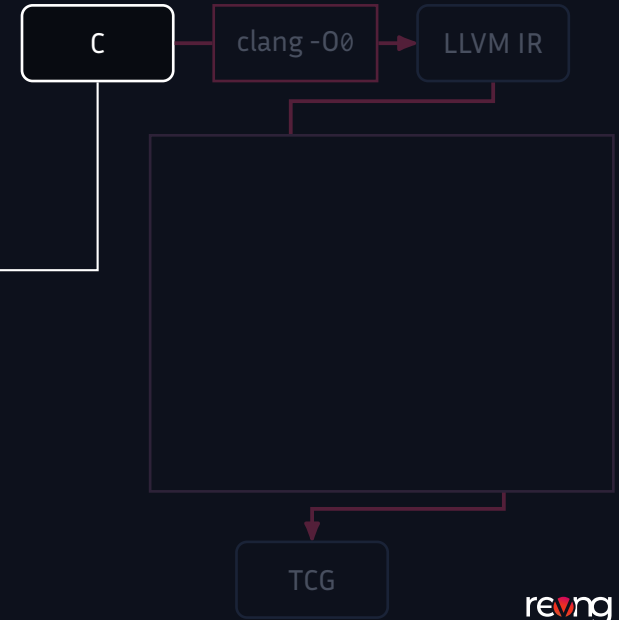
```
define i32 @predicated_add(i32 %dst, i32 %pred,
                           i32 %src1, i32 %src2) {
  %4 = alloca i32
  store i32 %dst, i32* %4

  %5 = and i32 %pred, 1
  %6 = icmp ne i32 %5, 0
  br i1 %6, label %if-label, label %exit-label

if-label:
  %8 = add i32 %src1, %src2
  store i32 %8, i32* %4
  br %exit-label

exit-label:
  %10 = load i32, i32* %4
  ret i32 %10
}
```

```
mov_i32 loc1, dst

and_i32 loc2, pred, $0x1
setcond_i32 loc3, pred, $0x0, ne
brcond_i32 loc3, $0x0, eq, $L0
br $L1

set_label $L0
add_i32 loc4, src1, src2
mov_i32 loc1, loc4
br $L1

set_label $L1
mov_i32 dst, loc1
```

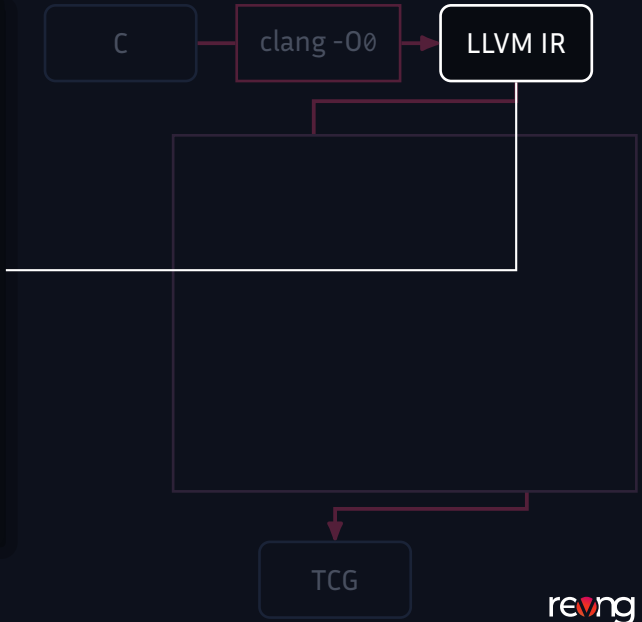
C


C
clang -O0
LLVM IR
helper2tcg
TCG

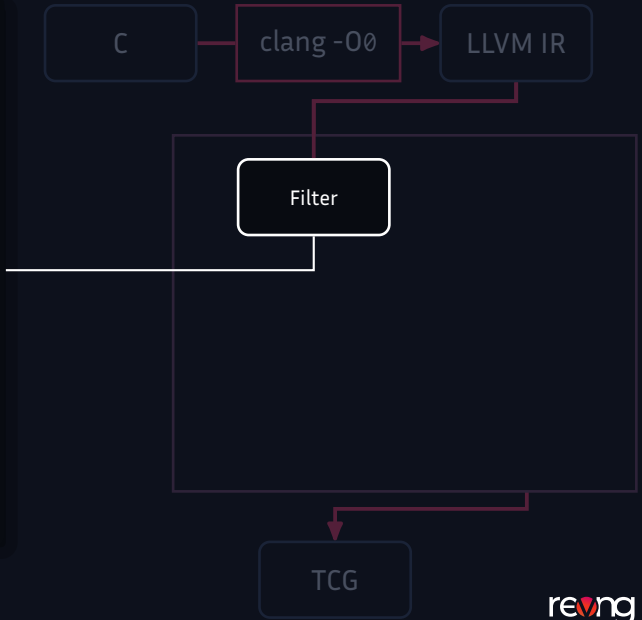
```c
#define CANCEL

int32_t A2_paddt(int32_t RdV,
                 int32_t PuV,
                 int32_t RsV,
                 int32_t RtV)
{
    if (PuV & 1) {
        RdV=RsV+RtV;
    } else {
        CANCEL;
    }

    return RdV;
}
```



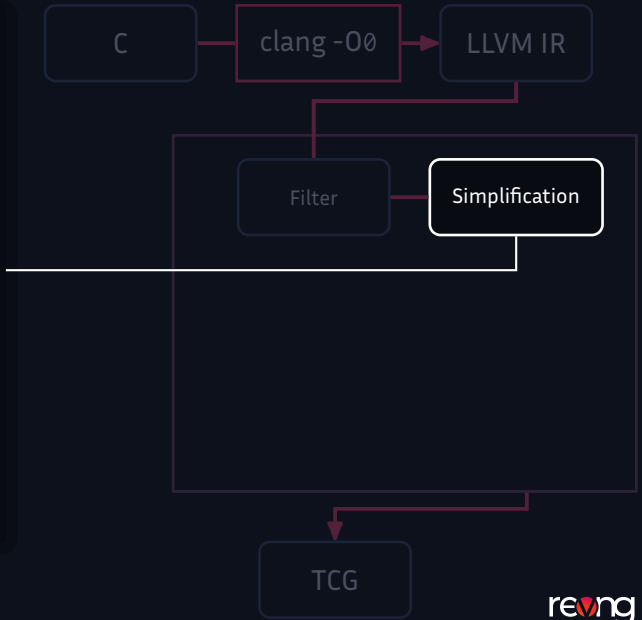
```
define i32 @A2_paddt(i32 %0, i32 %1, i32 %2, i32 %3) {
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  %7 = alloca i32, align 4
  %8 = alloca i32, align 4
  store i32 %0, i32* %5, align 4
  store i32 %1, i32* %6, align 4
  store i32 %2, i32* %7, align 4
  store i32 %3, i32* %8, align 4
  %9 = load i32, i32* %6, align 4
  %10 = and i32 %9, 1
  %11 = icmp ne i32 %10, 0
  br i1 %11, label %12, label %16

12:
  %13 = load i32, i32* %7, align 4
  %14 = load i32, i32* %8, align 4
  %15 = add nsw i32 %13, %14
  store i32 %15, i32* %5, align 4
  br label %17

16:
  br label %17

17:
  %18 = load i32, i32* %5, align 4
  ret i32 %18
}
```



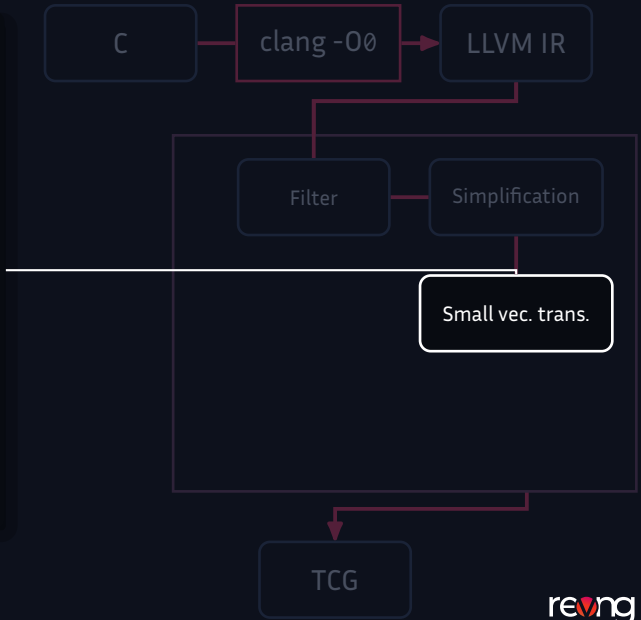
```
define i32 @A2_paddt(i32 %0, i32 %1, i32 %2, i32 %3) {
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  %7 = alloca i32, align 4
  %8 = alloca i32, align 4
  store i32 %0, i32* %5, align 4
  store i32 %1, i32* %6, align 4
  store i32 %2, i32* %7, align 4
  store i32 %3, i32* %8, align 4
  %9 = load i32, i32* %6, align 4
  %10 = and i32 %9, 1
  %11 = icmp ne i32 %10, 0
  br i1 %11, label %12, label %16

12:
  %13 = load i32, i32* %7, align 4
  %14 = load i32, i32* %8, align 4
  %15 = add nsw i32 %13, %14
  store i32 %15, i32* %5, align 4
  br label %17

16:
  br label %17

17:
  %18 = load i32, i32* %5, align 4
  ret i32 %18
}
```



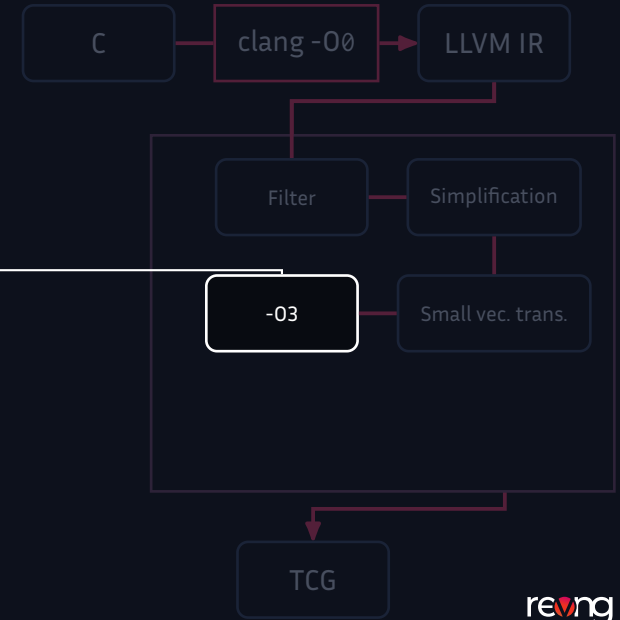
```
define i32 @A2_paddt(i32 %0, i32 %1,
                     i32 %2, i32 %3) {
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  store i32 %0, i32* %5, align 4
  store i32 %2, i32* %6, align 4
  %7 = and i32 %1, 1
  %.not = icmp eq i32 %7, 0
  br i1 %.not, label %11, label %8

8:
  %9 = load i32, i32* %6, align 4
  %10 = add nsw i32 %9, %3
  store i32 %10, i32* %5, align 4
  br label %11

11:
  %12 = load i32, i32* %5, align 4
  ret i32 %12
}
```



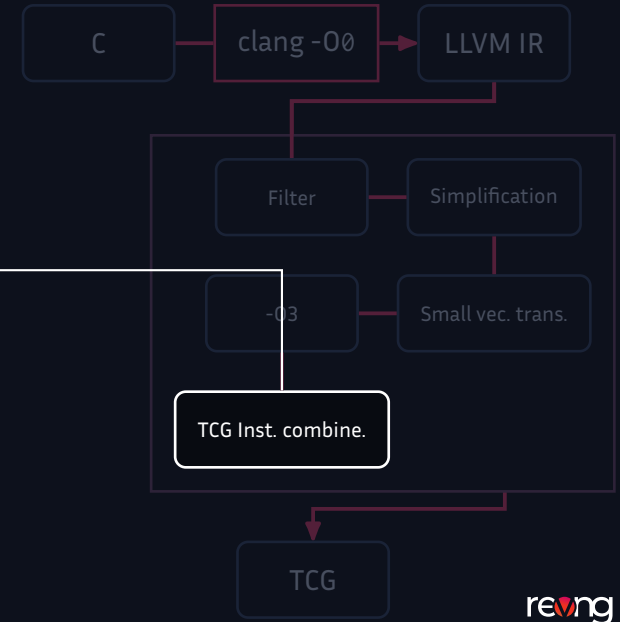
```
define i32 @A2_paddt(i32 %0, i32 %1,
                     i32 %2, i32 %3) {
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  store i32 %0, i32* %5, align 4
  store i32 %2, i32* %6, align 4
  %7 = and i32 %1, 1
  %.not = icmp eq i32 %7, 0
  br i1 %.not, label %10, label %8

8:
  %9 = add nsw i32 %2, %3
  store i32 %9, i32* %5, align 4
  br label %10

10:
  %11 = load i32, i32* %5, align 4
  ret i32 %11
}
```



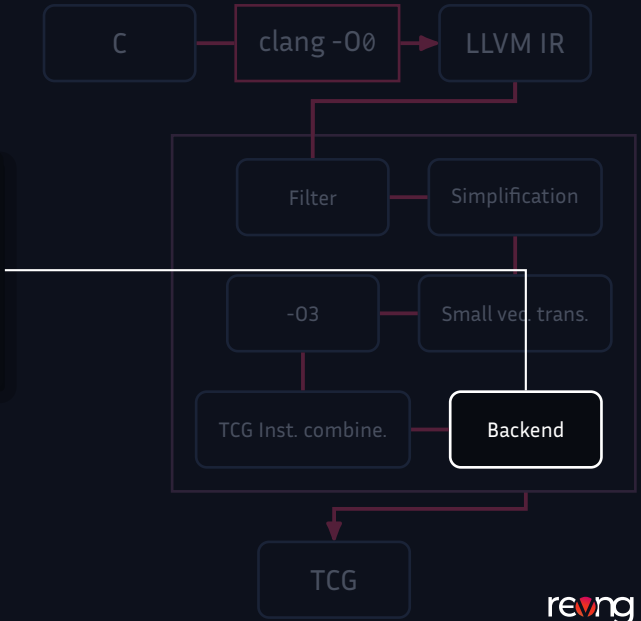
```
define i32 @A2_paddt(i32 %0, i32 %1,
                     i32 %2, i32 %3) {
  %5 = and i32 %1, 1
  %.not = icmp eq i32 %5, 0
  %6 = add nsw i32 %3, %2
  %spec.select = select i1 %.not, i32 %0,
                                  i32 %6
  ret i32 %spec.select
}
```



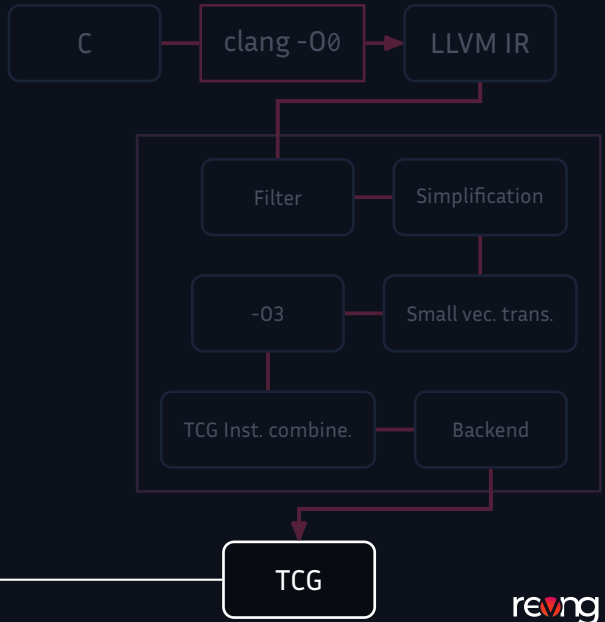
```
define i32 @A2_paddt(i32 %0, i32 %1,
                     i32 %2, i32 %3) {
  %5 = and i32 %1, 1
  %6 = add nsw i32 %3, %2
  %7 = call i32 @movcond.eq.i32(i32 %5,
                  i32 0, i32 %0, i32 %6)
  ret i32 %7
}
```



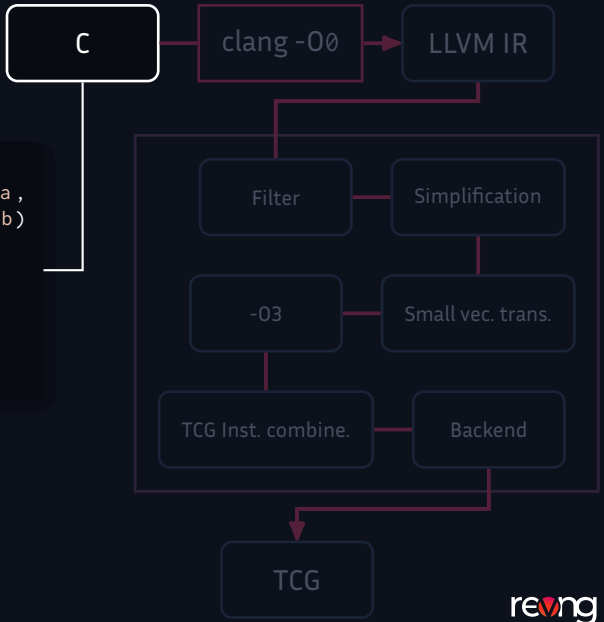
```
define i32 @A2_paddt(i32 %0, i32 %1,
                     i32 %2, i32 %3) {
  %5 = and i32 %1, 1
  %6 = add nsw i32 %3, %2
  %7 = call i32 @movcond.eq.i32(i32 %5,
                  i32 0, i32 %0, i32 %6)
  ret i32 %7
}
```



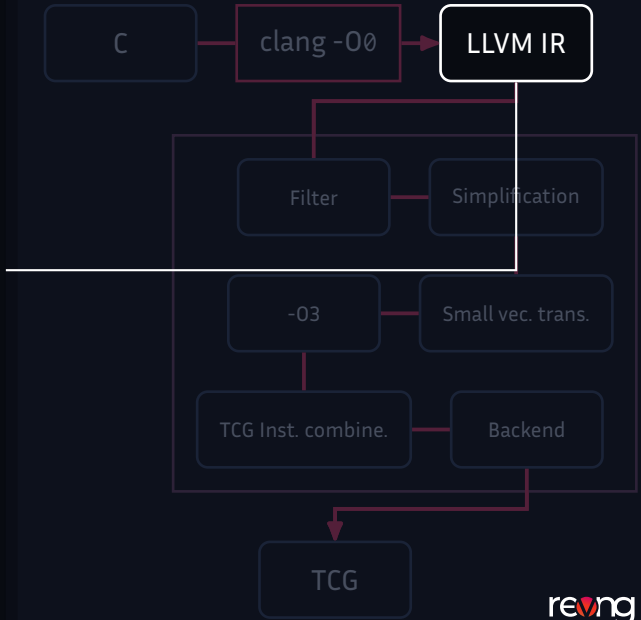
```
void A2_paddt(TCGv_i32 ret,
              TCGv_i32 v_0,
              TCGv_i32 v_1,
              TCGv_i32 v_2,
              TCGv_i32 v_3)
{
    tcg_gen_andi_i32(ret, v_0, 1);
    TCGv_i32 add_3 = tcg_temp_new_i32();
    tcg_gen_add_i32(add_3, v_3, v_2);
    tcg_gen_movcond_i32(TCG_COND_EQ, ret,
                ret, tcg_constant_i32(0),
                v_1, add_3);
}
```



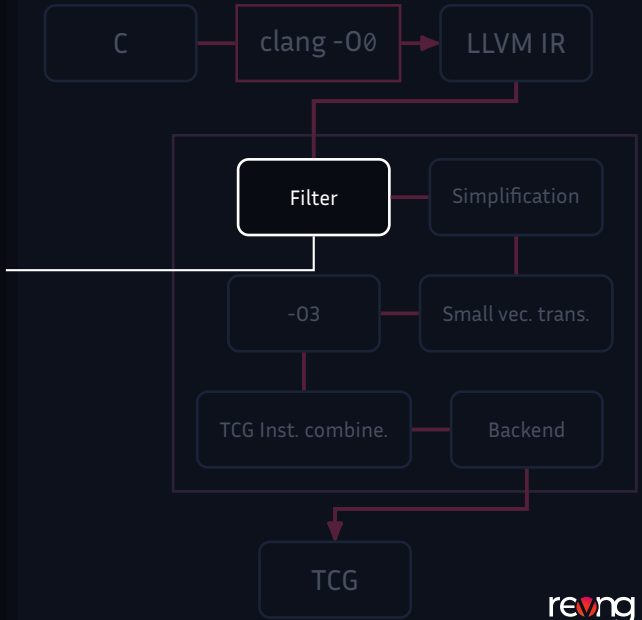
```c
void vec_add_bytes(uint8_t *dst_vec,
                   uint8_t *src_vec_a,
                   uint8_t *src_vec_b)
{
    for (int i = 0; i < 128; ++i) {
        dst_vec[i] = src_vec_a[i]
                   + src_vec_b[i];
    }
}
```



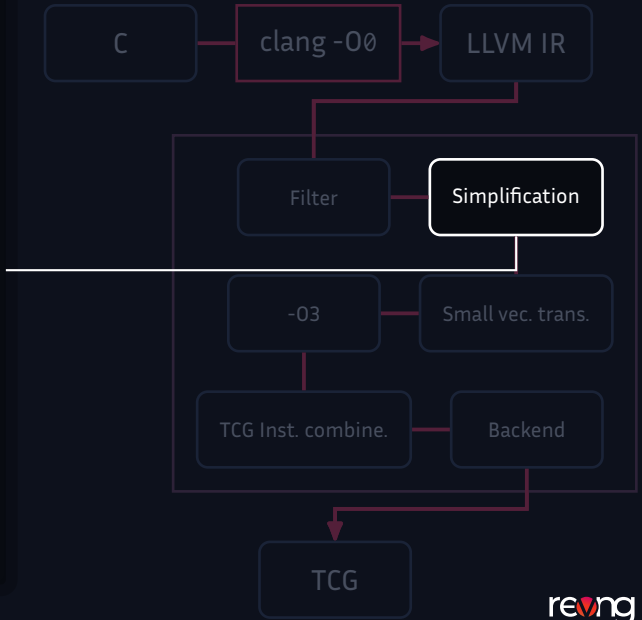
```
  %11 = load i8*, i8** %4, align 8
  store i8* %11, i8** %7, align 8
  %12 = load i8*, i8** %5, align 8
  store i8* %12, i8** %8, align 8
  %13 = load i8*, i8** %6, align 8
  store i8* %13, i8** %9, align 8
  store i32 0, i32* %10, align 4
  br label %14

14:
  %15 = load i32, i32* %10, align 4
  %16 = icmp slt i32 %15, 128
  br i1 %16, label %17, label %39

17:
  %18 = load i8*, i8** %8, align 8
  %19 = load i32, i32* %10, align 4
  %20 = sext i32 %19 to i64
  %21 = getelementptr inbounds i8, i8* %18, i64 %20
  %22 = load i8, i8* %21, align 1
  %23 = zext i8 %22 to i32
  %24 = load i8*, i8** %9, align 8
  %25 = load i32, i32* %10, align 4
  %26 = sext i32 %25 to i64
  %27 = getelementptr inbounds i8, i8* %24, i64 %26
  %28 = load i8, i8* %27, align 1
  %29 = zext i8 %28 to i32
  %30 = add nsw i32 %23, %29
  %31 = trunc i32 %30 to i8
  %32 = load i8*, i8** %7, align 8
  %33 = load i32, i32* %10, align 4
  %34 = sext i32 %33 to i64
  %35 = getelementptr inbounds i8, i8* %32, i64 %34
  store i8 %31, i8* %35, align 1
  br label %36

36:
```



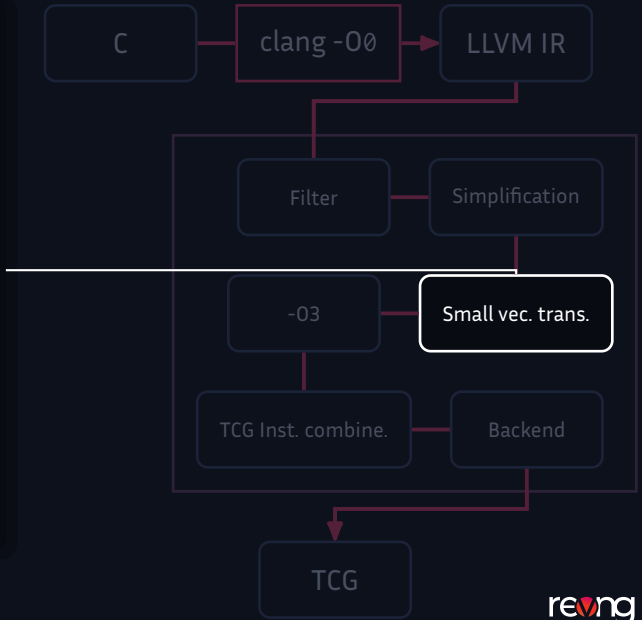
```
  %11 = load i8*, i8** %4, align 8
  store i8* %11, i8** %7, align 8
  %12 = load i8*, i8** %5, align 8
  store i8* %12, i8** %8, align 8
  %13 = load i8*, i8** %6, align 8
  store i8* %13, i8** %9, align 8
  store i32 0, i32* %10, align 4
  br label %14

14:
  %15 = load i32, i32* %10, align 4
  %16 = icmp slt i32 %15, 128
  br i1 %16, label %17, label %39

17:
  %18 = load i8*, i8** %8, align 8
  %19 = load i32, i32* %10, align 4
  %20 = sext i32 %19 to i64
  %21 = getelementptr inbounds i8, i8* %18, i64 %20
  %22 = load i8, i8* %21, align 1
  %23 = zext i8 %22 to i32
  %24 = load i8*, i8** %9, align 8
  %25 = load i32, i32* %10, align 4
  %26 = sext i32 %25 to i64
  %27 = getelementptr inbounds i8, i8* %24, i64 %26
  %28 = load i8, i8* %27, align 1
  %29 = zext i8 %28 to i32
  %30 = add nsw i32 %23, %29
  %31 = trunc i32 %30 to i8
  %32 = load i8*, i8** %7, align 8
  %33 = load i32, i32* %10, align 4
  %34 = sext i32 %33 to i64
  %35 = getelementptr inbounds i8, i8* %32, i64 %34
  store i8 %31, i8* %35, align 1
  br label %36

36:
```



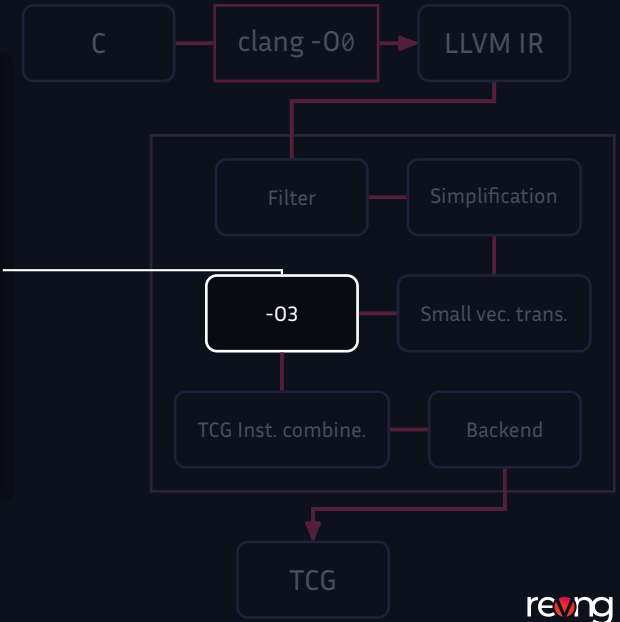
```
define void @vec_add_bytes(i8* %0, i8* %1, i8* %2) {
  %4 = alloca i8*, align 8
  %5 = alloca i8*, align 8
  %6 = alloca i8*, align 8
  store i8* %0, i8** %4, align 8
  store i8* %1, i8** %5, align 8
  store i8* %2, i8** %6, align 8
  br label %7

7:
  %storemerge = phi i32 [ 0, %3 ], [ %20, %9 ]
  %8 = icmp ult i32 %storemerge, 128
  br i1 %8, label %9, label %21

9:
  %10 = load i8*, i8** %5, align 8
  %11 = zext i32 %storemerge to i64
  %12 = getelementptr inbounds i8, i8* %10, i64 %11
  %13 = load i8, i8* %12, align 1
  %14 = load i8*, i8** %6, align 8
  %15 = getelementptr inbounds i8, i8* %14, i64 %11
  %16 = load i8, i8* %15, align 1
  %17 = add i8 %16, %13
  %18 = load i8*, i8** %4, align 8
  %19 = getelementptr inbounds i8, i8* %18, i64 %11
  store i8 %17, i8* %19, align 1
  %20 = add nuw nsw i32 %storemerge, 1
  br label %7, !llvm.loop !10

21:
  ret void
}
```



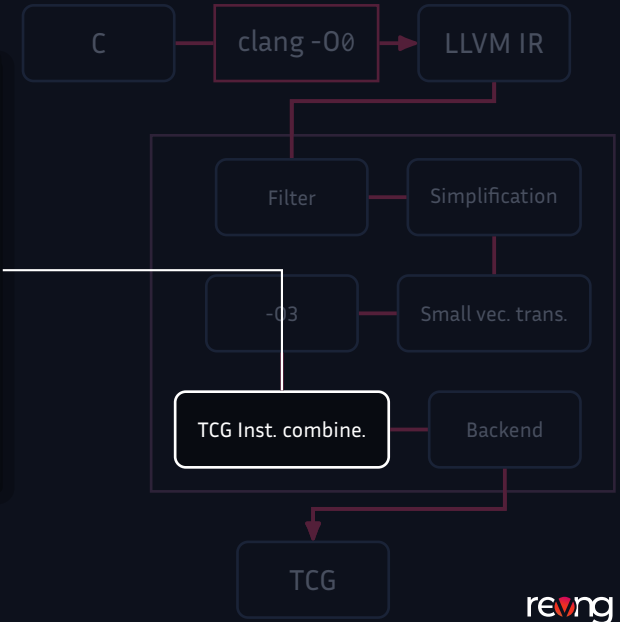
```
define void @vec_add_bytes(i8* %0, i8* %1, i8* %2) {
  %4 = alloca i8*, align 8
  %5 = alloca i8*, align 8
  %6 = alloca i8*, align 8
  store i8* %0, i8** %4, align 8
  store i8* %1, i8** %5, align 8
  store i8* %2, i8** %6, align 8
  br label %7

7:
  %iv2 = phi i64 [ 0, %3 ], [ %iv.next, %7 ]
  %8 = load i8*, i8** %5, align 8
  %9 = getelementptr inbounds i8, i8* %8, i64 %iv2
  %10 = load i8, i8* %9, align 1
  %11 = load i8*, i8** %6, align 8
  %12 = getelementptr inbounds i8, i8* %11, i64 %iv2
  %13 = load i8, i8* %12, align 1
  %14 = add i8 %13, %10
  %15 = load i8*, i8** %4, align 8
  %16 = getelementptr inbounds i8, i8* %15, i64 %iv2
  store i8 %14, i8* %16, align 1
  %iv.next = add nuw nsw i64 %iv2, 1
  %exitcond = icmp ne i64 %iv.next, 128
  br i1 %exitcond, label %7, label %17, !llvm.loop !10

17:
  ret void
}
```



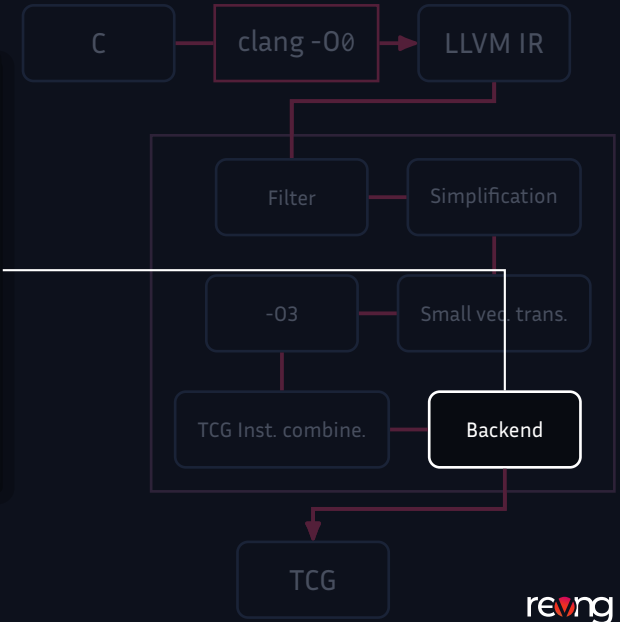
```
define void @vec_add_bytes(i8* %0,
                           i8* %1,
                           i8* %2) {
  %3 = bitcast i8* %1 to <128 x i8>*
  %load = load <128 x i8>, <128 x i8>* %3

  %4 = bitcast i8* %2 to <128 x i8>*
  %load5 = load <128 x i8>, <128 x i8>* %4

  %5 = add <128 x i8> %load5, %load

  %6 = bitcast i8* %0 to <128 x i8>*
  store <128 x i8> %5, <128 x i8>* %6

  ret void
}
```



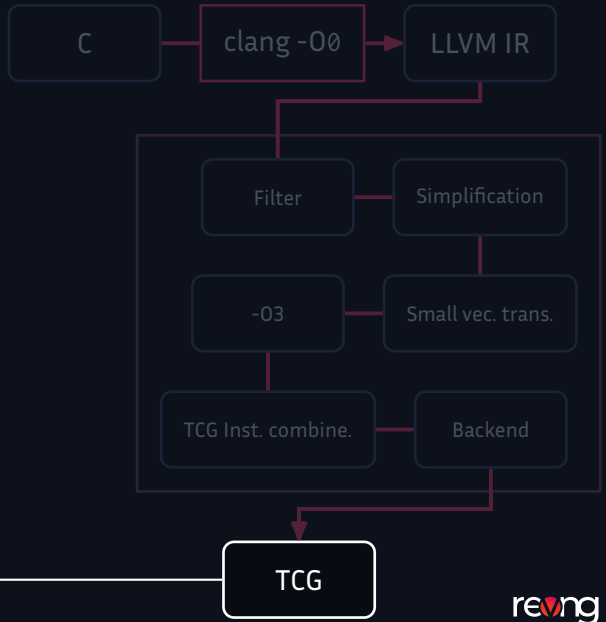
```
define void @vec_add_bytes(i8* %0,
                           i8* %1,
                           i8* %2) {
  %3 = bitcast i8* %1 to <128 x i8>*
  %load = load <128 x i8>, <128 x i8>* %3

  %4 = bitcast i8* %2 to <128 x i8>*
  %load5 = load <128 x i8>, <128 x i8>* %4

  %5 = add <128 x i8> %load5, %load

  %6 = bitcast i8* %0 to <128 x i8>*
  store <128 x i8> %5, <128 x i8>* %6

  ret void
}
```



```
define void @vec_add_bytes(i8* %0,
                           i8* %1,
                           i8* %2) {
  %3 = bitcast i8* %1 to <128 x i8>*
  %load = load <128 x i8>, <128 x i8>* %3

  %4 = bitcast i8* %2 to <128 x i8>*
  %load5 = load <128 x i8>, <128 x i8>* %4

  %5 = add <128 x i8> %load5, %load

  %6 = bitcast i8* %0 to <128 x i8>*
  store <128 x i8> %5, <128 x i8>* %6

  ret void
}
```



```
void vec_add_bytes(intptr_t vo_0,
                   intptr_t vo_1,
                   intptr_t vo_2) {
    tcg_gen_gvec_add(MO_8,
                     vo_0, vo_2, vo_1,
                     128, 128);
}
```



Background (Hexagon & QEMU)

idef-parser

helper2tcg

Demo

Benchmarks (Hexagon)

Summary

Background (Hexagon & QEMU)

idef-parser

helper2tcg

Demo

Benchmarks (Hexagon)

Summary

- **Motivation**: Compare different instruction implementations

- **Motivation**: Compare different instruction implementations
- General purpose benchmarks (speedup relative to helpers + manual TCG)

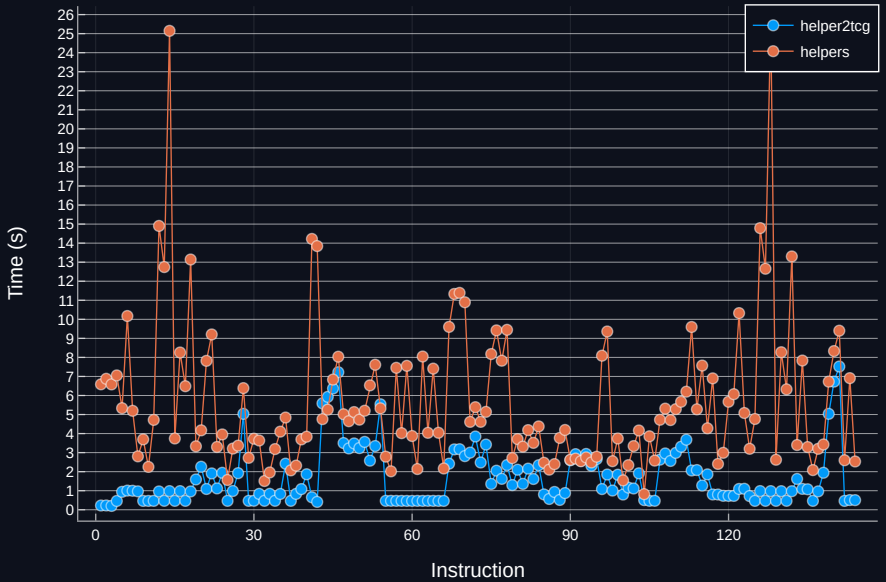
- **Motivation**: Compare different instruction implementations
- General purpose benchmarks (speedup relative to helpers + manual TCG)

Bench.	idef-parser	helper2tcg
coremark	2.29x	2.43x
stream	1.52x	1.60x
nbench	2.85x	3.09x

- runtime instruction count reduction (%) relative to helpers + manual TCG

- runtime instruction count reduction (%) relative to helpers + manual TCG

Binary	idef-parser	helper2tcg
nn tests	3.3%	9.6%
corner det.	1.4%	11%


helper2tcg
helpers
Time (s)
Instruction
0
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
0
30
60
90
120

- Average speedup 4.30x
- Peak speedup 33.6x

Background (Hexagon & QEMU)

idef-parser

helper2tcg

Demo

Benchmarks (Hexagon)

Summary

- helper2tcg:
  - Build-time tool using clang and LLVM
  - Helper-function $\rightarrow$ TCG translation
  - Architecture independent

- helper2tcg:
  - Build-time tool using clang and LLVM
  - Helper-function $\rightarrow$ TCG translation
  - Architecture independent
- Applied to Hexagon:
  - Emit TCG for 1400 instructions
  - Able to entirely replace idef-parser
  - Emits more efficient/smaller TCG compared to idef-parser
  - Handles more complicated instructions

- helper2tcg:
  - Build-time tool using clang and LLVM
  - Helper-function $\rightarrow$ TCG translation
  - Architecture independent
- Applied to Hexagon:
  - Emit TCG for 1400 instructions
  - Able to entirely replace idef-parser
  - Emits more efficient/smaller TCG compared to idef-parser
  - Handles more complicated instructions
- Plan to open source/upstream :)

Get in touch at:
info@rev.ng

Subscribe to our newsletter:
https://rev.ng/newsletter.html

For each instruction, generate test stressing that instruction

For each instruction, generate test stressing that instruction

```
r0 = add(r1,r2)
```

For each instruction, generate test stressing that instruction

```
r0 = add(r1,r2)
```

```
_start:
    // Loop prologue / setup
    [...]

loop:
    // Loop iteration
    r2=add(r3,r4)
    r3=add(r2,r4)
    r2=add(r3,r4)
    r3=add(r2,r4)
    [...]
    r2=add(r3,r4)
    r3=add(r2,r4)

    // Loop epilogue
    r0 = add(r0, #1)
    p0 = cmp.lt(r0, r1)
    if (p0) jump loop

    // Exit
    [...]
```