Rust JIT demo

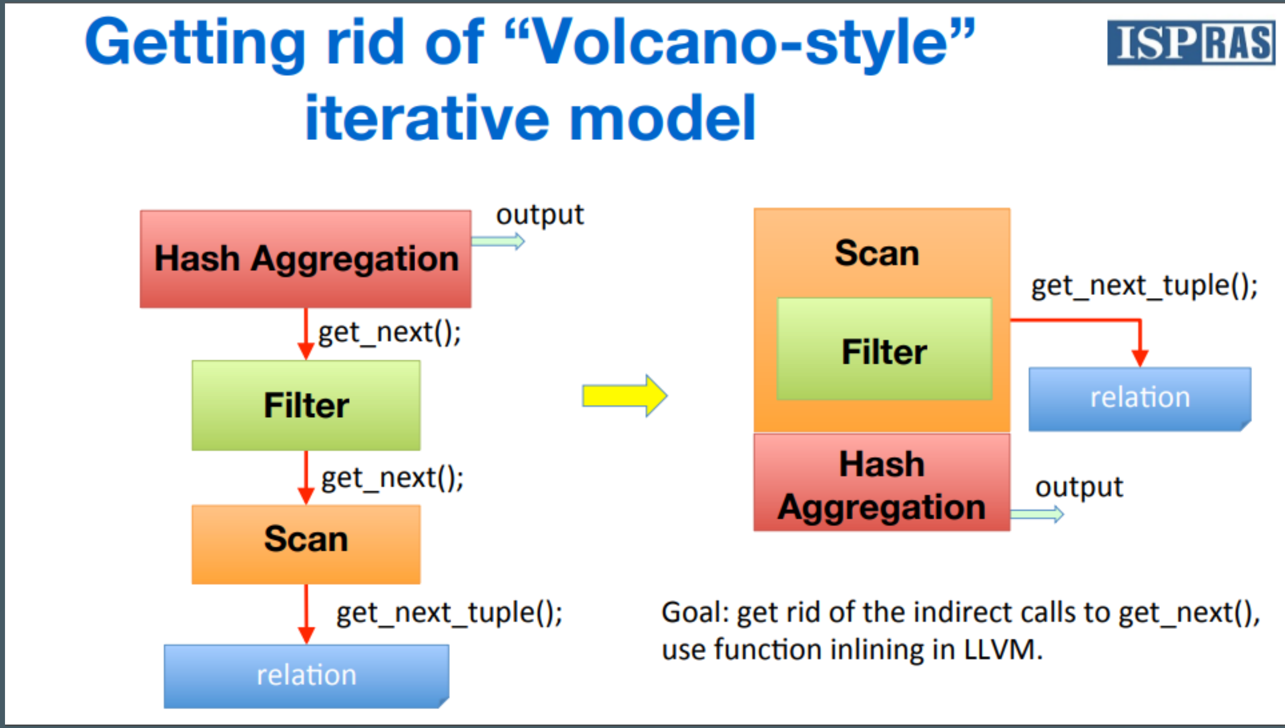
Making your code blazingly faster (kinda)

Why use JIT anyway?

- Most Rust code doesn't need JIT.
- JIT is useful for:
  - Dynamic algorithms/pipelines (e.g. SQL queries)
  - Dynamic execution (e.g. console emulators)
  - Basically anything highly dynamic

JIT case study

A common success story of adding LLVM JIT is PostgreSQL. Here's a link to the slides of one of their presentations: [Slides]


Getting rid of "Volcano-style" iterative model

Goal: get rid of the indirect calls to get_next(), use function inlining in LLVM.

How do we JIT Rust?

After a LOT of trial and error, I decided PostgreSQL's approach is best.

It involves:

- Converting constant code modules to raw LLVM IR
- Linking the pre-compiled functions at runtime in desired ways.

They do it in C/C++, but it's possible in Rust as well.

So what's the process?

In theory it's simple really.

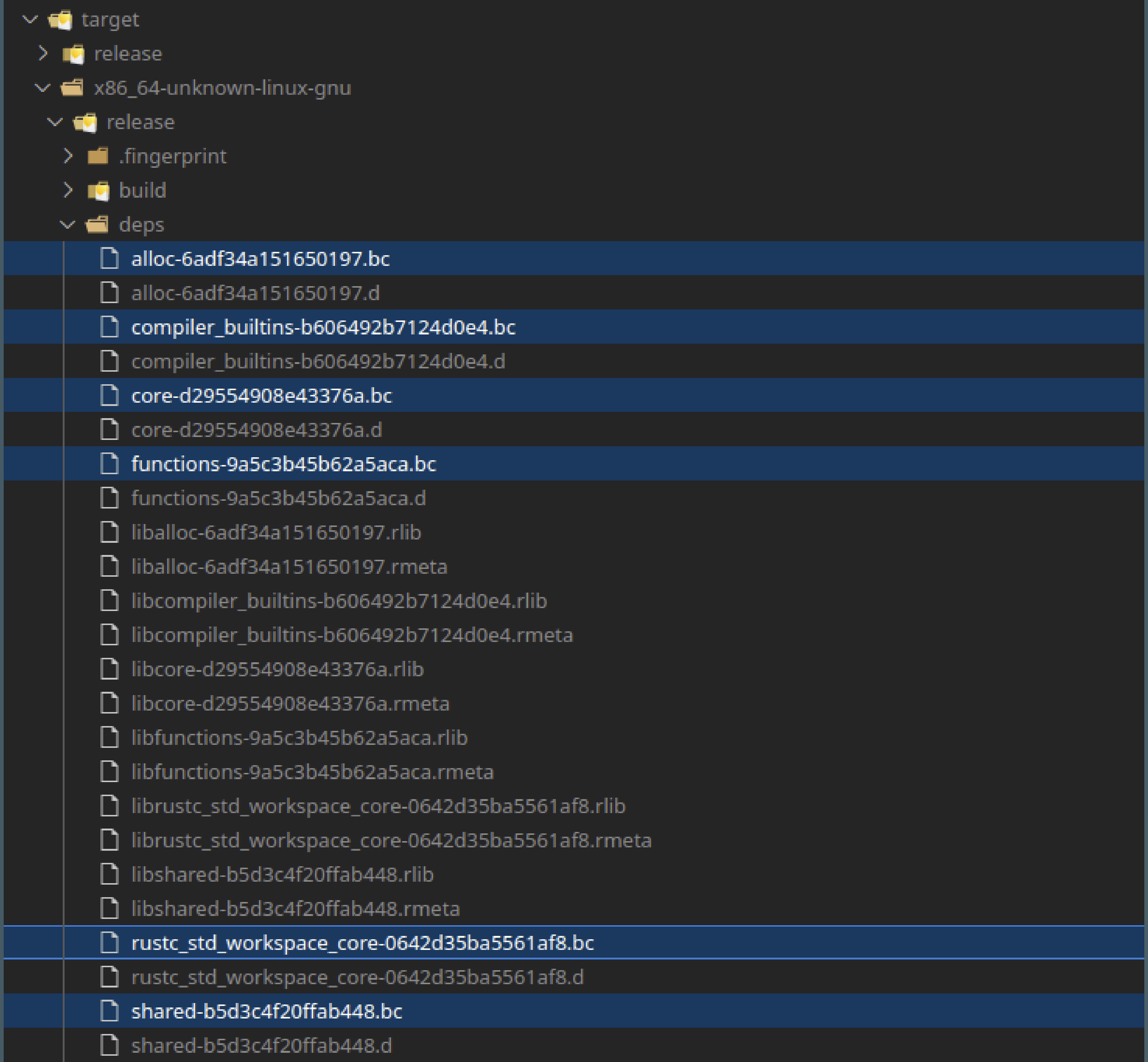
1. Convert Rust code to LLVM IR
2. Use the pre-compiled functions at runtime to build new code to JIT
3. Cry (`[1] 68684 segmentation fault`)
4. Run the JIT code
5. Profit

Step 1: Convert Rust code to LLVM IR

rustc provides a flag called `--emit=llvm-bc`, which emits the LLVM bytecode.

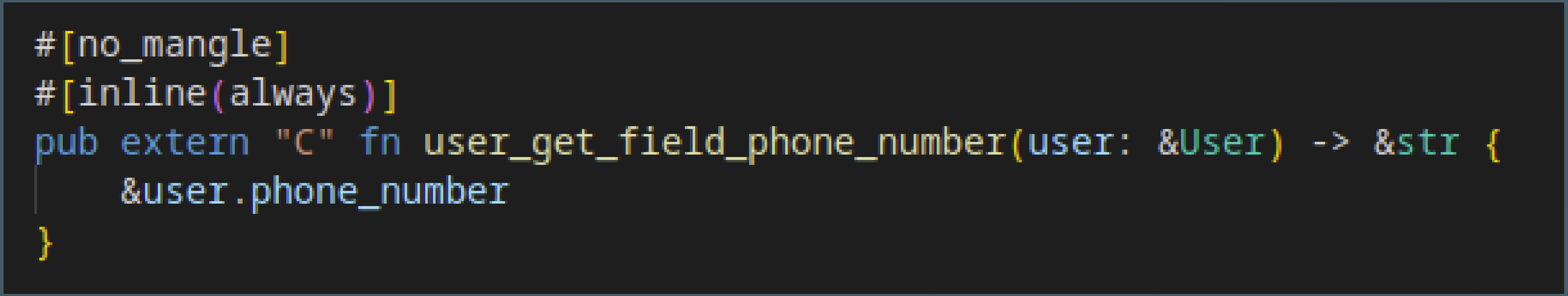
This returns us the LLVM IR for each compiled module (individually, unlinked).

You can also compile with `--emit=llvm-ir` to get human readable LLVM IR.

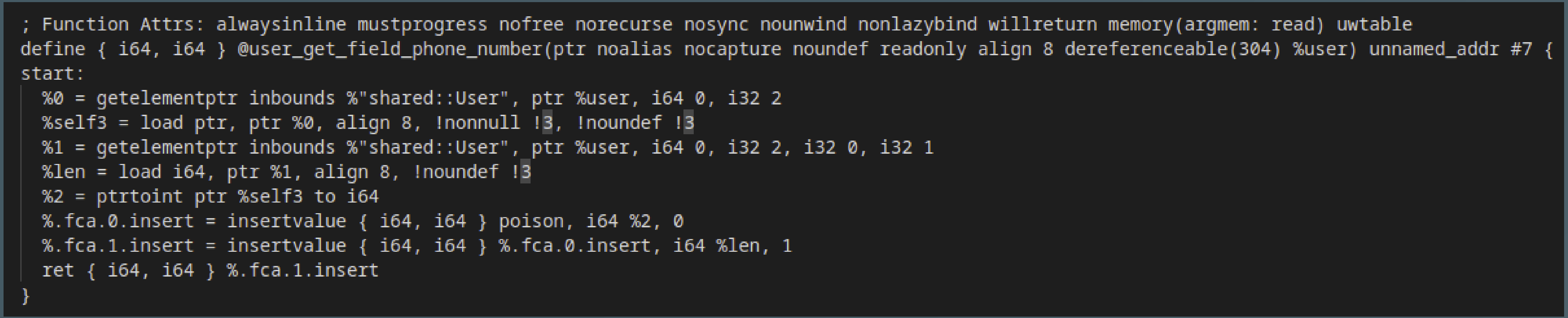


```
target
  release
  x86_64-unknown-linux-gnu
    release
      .fingerprint
      build
      deps
        alloc-6adf34a151650197.bc
        alloc-6adf34a151650197.d
        compiler_builtins-b606492b7124d0e4.bc
        compiler_builtins-b606492b7124d0e4.d
        core-d29554908e43376a.bc
        core-d29554908e43376a.d
        functions-9a5c3b45b62a5aca.bc
        functions-9a5c3b45b62a5aca.d
        liballoc-6adf34a151650197.rlib
        liballoc-6adf34a151650197.rmeta
        libcompiler_builtins-b606492b7124d0e4.rlib
        libcompiler_builtins-b606492b7124d0e4.rmeta
        libcore-d29554908e43376a.rlib
        libcore-d29554908e43376a.rmeta
        libfunctions-9a5c3b45b62a5aca.rlib
        libfunctions-9a5c3b45b62a5aca.rmeta
        librustc_std_workspace_core-0642d35ba5561af8.rlib
        librustc_std_workspace_core-0642d35ba5561af8.rmeta
        libshared-b5d3c4f20ffab448.rlib
        libshared-b5d3c4f20ffab448.rmeta
        rustc_std_workspace_core-0642d35ba5561af8.bc
        rustc_std_workspace_core-0642d35ba5561af8.d
        shared-b5d3c4f20ffab448.bc
        shared-b5d3c4f20ffab448.d
```

Example Rust to LLVM transformation



```rust
#[no_mangle]
#[inline(always)]
pub extern "C" fn user_get_field_phone_number(user: &User) -> &str {
    &user.phone_number
}
```



```llvm
; Function Attrs: alwaysinline mustprogress nofree norecurse nosync nounwind nonlazybind willreturn memory(argmem: read) uwtable
define { i64, i64 } @user_get_field_phone_number(ptr noalias nocapture noundef readonly align 8 dereferenceable(304) %user) unnamed_addr #7 {
start:
  %0 = getelementptr inbounds %"shared::User", ptr %user, i64 0, i32 2
  %self3 = load ptr, ptr %0, align 8, !nonnull !3, !noundef !3
  %1 = getelementptr inbounds %"shared::User", ptr %user, i64 0, i32 2, i32 0, i32 1
  %len = load i64, ptr %1, align 8, !noundef !3
  %2 = ptrtoint ptr %self3 to i64
  %.fca.0.insert = insertvalue { i64, i64 } poison, i64 %2, 0
  %.fca.1.insert = insertvalue { i64, i64 } %.fca.0.insert, i64 %len, 1
  ret { i64, i64 } %.fca.1.insert
}
```

But it has undeclared functions...

```
; core::panicking::panic
; Function Attrs: cold noinline noreturn nonlazybind uwtable
declare void @_ZN4core9panicking5panic17hde99db60e42ff4f8E(ptr noalias noundef nonnull readonly align 1, i64 noundef, ptr noalias noundef rea

; <alloc::string::String as core::clone::Clone>::clone
; Function Attrs: nonlazybind uwtable
declare void @"_ZN60_$LT$alloc..string..String$u20$as$u20$core..clone..Clone$GT$5clone17h3f04fa797a421f73E"(ptr noalias nocapture noundef sre

; core::slice::memchr::memchr_aligned
; Function Attrs: nonlazybind uwtable
declare { i64, i64 } @_ZN4core5slice6memchr14memchr_aligned17h64bbfb14ca74da7dE(i8 noundef, ptr noalias noundef nonnull readonly align 1, i6

; core::str::pattern::StrSearcher::new
; Function Attrs: nonlazybind uwtable
declare void @_ZN4core3str7pattern11StrSearcher3new17h80062ab20431da3dE(ptr noalias nocapture noundef sret(%"core::str::pattern::StrSearcher
```

Step 1.5: Linking them all together

TL;DR: Use LLVM to link the IR together into one module.

This is harder than it sounds, because a lot of manipulation is required here.

But a working version can be found in `./compile`.

Step 1.5.5: Including `core`

There are still some missing functions, because `core` needs to be compiled to IR for linking too.

This can be done by using `-Z build-std="core,alloc"`, which requires nightly rust.

More undeclared functions...

There's still functions like `__rust_alloc` and `rust_begin_unwind`.

These can be linked in at runtime, as they closely relate to the executable.

Step 1.6: Optimize

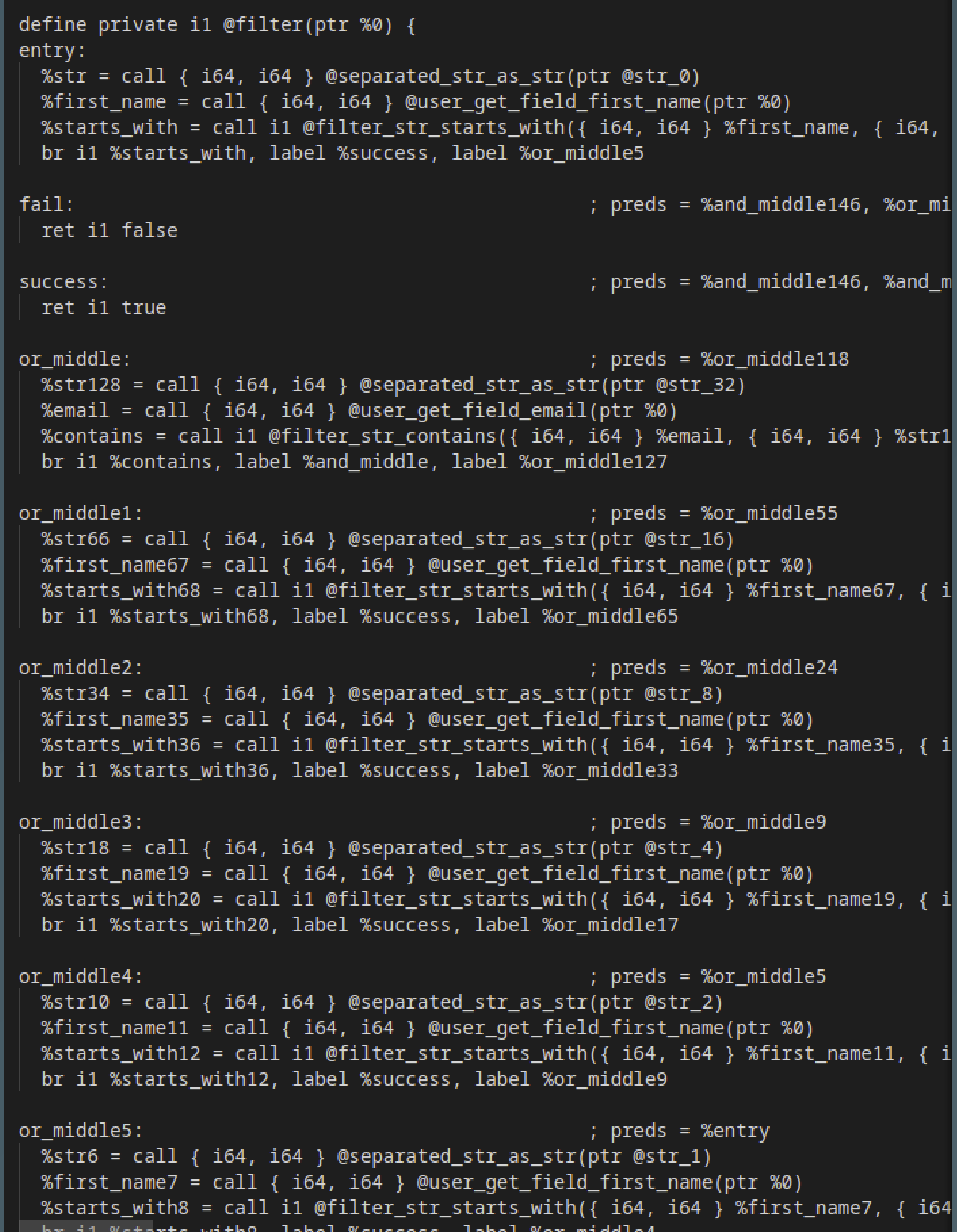
Run optimization passes on the linked IR.

Keep control over what's marked public and what's private, so:

- Private stuff gets inlined more
- Unused private functions get optimized away.

Step 2 (finally)

Importing these functions, and joining them together!



```
define private i1 @filter(ptr %0) {
entry:
  %str = call { i64, i64 } @separated_str_as_str(ptr @str_0)
  %first_name = call { i64, i64 } @user_get_field_first_name(ptr %0)
  %starts_with = call i1 @filter_str_starts_with({ i64, i64 } %first_name, { i64,
  br i1 %starts_with, label %success, label %or_middle5

fail:                                             ; preds = %and_middle146, %or_mi
  ret i1 false

success:                                          ; preds = %and_middle146, %and_m
  ret i1 true

or_middle:                                        ; preds = %or_middle118
  %str128 = call { i64, i64 } @separated_str_as_str(ptr @str_32)
  %email = call { i64, i64 } @user_get_field_email(ptr %0)
  %contains = call i1 @filter_str_contains({ i64, i64 } %email, { i64, i64 } %str1
  br i1 %contains, label %and_middle, label %or_middle127

or_middle1:                                       ; preds = %or_middle55
  %str66 = call { i64, i64 } @separated_str_as_str(ptr @str_16)
  %first_name67 = call { i64, i64 } @user_get_field_first_name(ptr %0)
  %starts_with68 = call i1 @filter_str_starts_with({ i64, i64 } %first_name67, { i
  br i1 %starts_with68, label %success, label %or_middle65

or_middle2:                                       ; preds = %or_middle24
  %str34 = call { i64, i64 } @separated_str_as_str(ptr @str_8)
  %first_name35 = call { i64, i64 } @user_get_field_first_name(ptr %0)
  %starts_with36 = call i1 @filter_str_starts_with({ i64, i64 } %first_name35, { i
  br i1 %starts_with36, label %success, label %or_middle33

or_middle3:                                       ; preds = %or_middle9
  %str18 = call { i64, i64 } @separated_str_as_str(ptr @str_4)
  %first_name19 = call { i64, i64 } @user_get_field_first_name(ptr %0)
  %starts_with20 = call i1 @filter_str_starts_with({ i64, i64 } %first_name19, { i
  br i1 %starts_with20, label %success, label %or_middle17

or_middle4:                                       ; preds = %or_middle5
  %str10 = call { i64, i64 } @separated_str_as_str(ptr @str_2)
  %first_name11 = call { i64, i64 } @user_get_field_first_name(ptr %0)
  %starts_with12 = call i1 @filter_str_starts_with({ i64, i64 } %first_name11, { i
  br i1 %starts_with12, label %success, label %or_middle9

or_middle5:                                       ; preds = %entry
  %str6 = call { i64, i64 } @separated_str_as_str(ptr @str_1)
  %first_name7 = call { i64, i64 } @user_get_field_first_name(ptr %0)
  %starts_with8 = call i1 @filter_str_starts_with({ i64, i64 } %first_name7, { i64
  br i1 %starts_with8, label %success, label %or_middle4
```

A note on Rust ABI

The Rust ABI is not stable, and can act in unexpected ways. For example:

```rust
#[no_mangle]
pub extern "C" fn foo() -> Vec<u8> {
    Vec::new()
}
```

compiles to:

```llvm
define private void @foo(ptr (%"alloc::vec::Vec<u8>") %_0) { ; I've omitted some attributes
start:
  ; [Write to the pointer]
  ret void
}
```

Step 3: Misery

Anyone who had the misfortune of dealing with LLVM will know.

Get ready for a lot of `[1] 68684 segmentation fault`.

`rust-lldb` is your best friend.

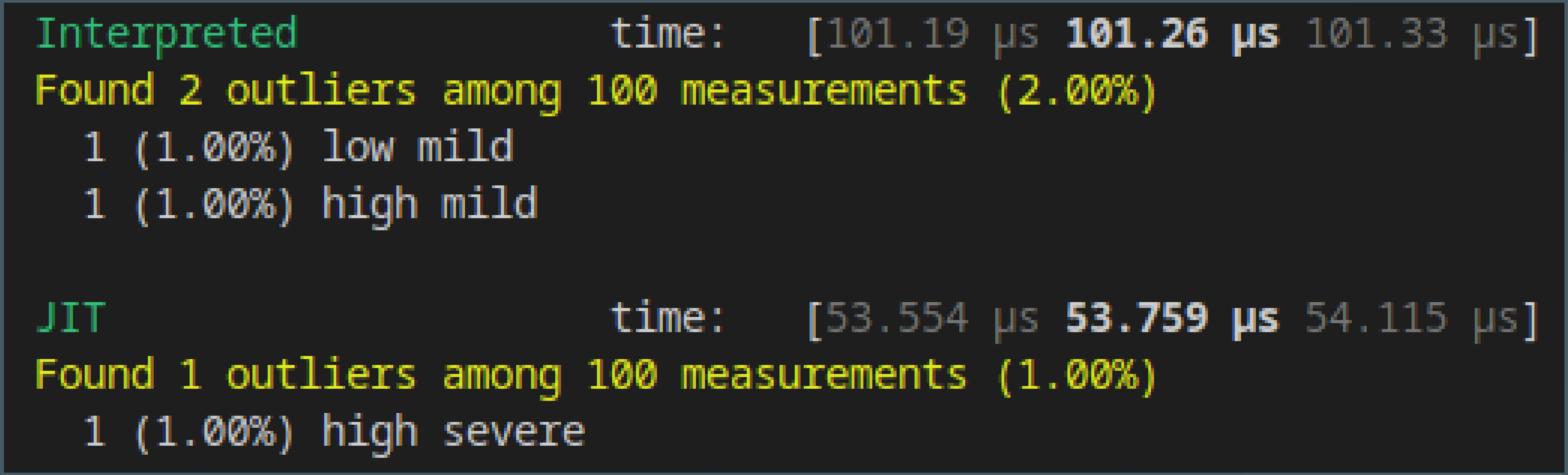
Also the LLVM discord server where some some American will answer your questions at 4am.

Step 4: Running it!

It took me a week to figure out how to use ORC JIT (via the American I mentioned).

But, it's seamless when it actually works.

Step 5: Profit



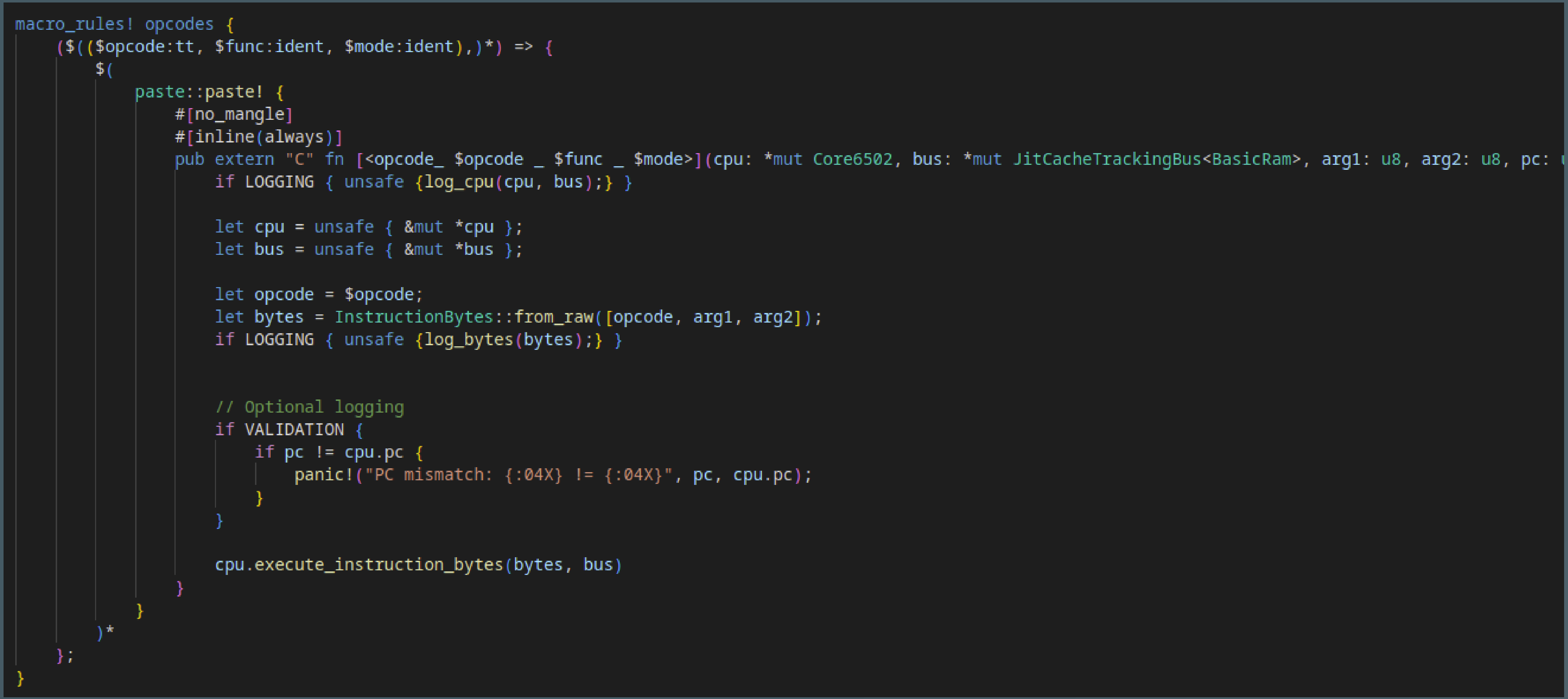
```
Interpreted                 time:   [101.19 µs 101.26 µs 101.33 µs]
Found 2 outliers among 100 measurements (2.00%)
  1 (1.00%) low mild
  1 (1.00%) high mild

JIT                         time:   [53.554 µs 53.759 µs 54.115 µs]
Found 1 outliers among 100 measurements (1.00%)
  1 (1.00%) high severe
```

Bonus

Emulation!

JITting 6502 bytecode

The opcode macro



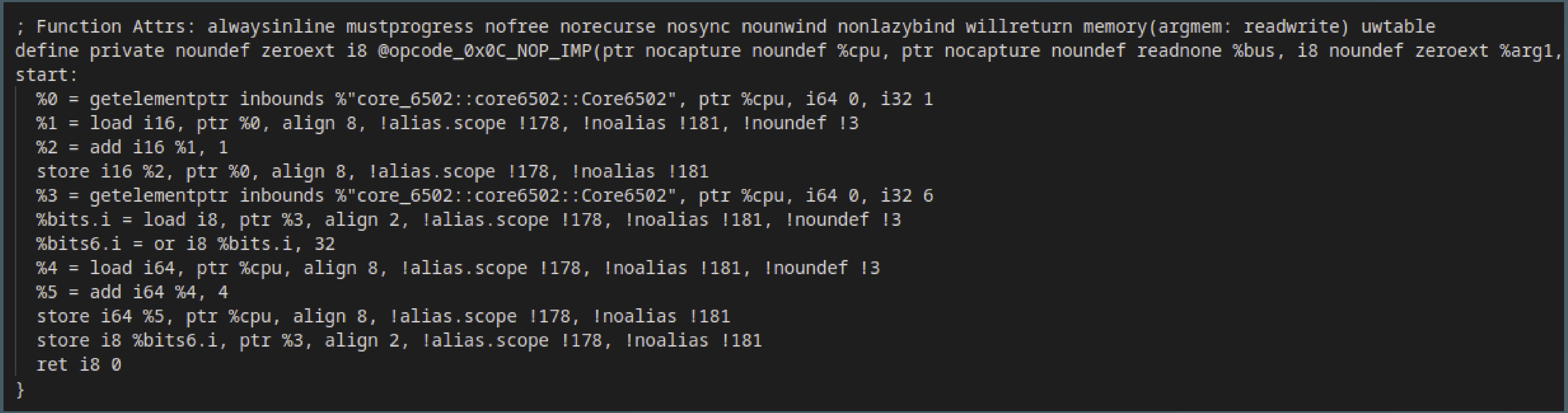
```rust
macro_rules! opcodes {
    ($(($opcode:tt, $func:ident, $mode:ident),)*) => {
        $(
            paste::paste! {
                #[no_mangle]
                #[inline(always)]
                pub extern "C" fn [<opcode_ $opcode _ $func _ $mode>](cpu: *mut Core6502, bus: *mut JitCacheTrackingBus<BasicRam>, arg1: u8, arg2: u8, pc: 
                    if LOGGING { unsafe {log_cpu(cpu, bus);} }

                    let cpu = unsafe { &mut *cpu };
                    let bus = unsafe { &mut *bus };

                    let opcode = $opcode;
                    let bytes = InstructionBytes::from_raw([opcode, arg1, arg2]);
                    if LOGGING { unsafe {log_bytes(bytes);} }


                    // Optional logging
                    if VALIDATION {
                        if pc != cpu.pc {
                            panic!("PC mismatch: {:04X} != {:04X}", pc, cpu.pc);
                        }
                    }

                    cpu.execute_instruction_bytes(bytes, bus)
                }
            }
        )*
    };
}
```

Opcode macro usage

```
opcodes! {
    (0x00, BRK, IMP), (0x01, ORA, IZX), (0x02, XXX, IMP), (0x03, XXX, IMP), (0x04, NOP, IMP), (0x05, ORA, ZP0), (0x06, ASL, ZP0), (0x07, XXX, IMP),
    (0x10, BPL, REL), (0x11, ORA, IZY), (0x12, XXX, IMP), (0x13, XXX, IMP), (0x14, NOP, IMP), (0x15, ORA, ZPX), (0x16, ASL, ZPX), (0x17, XXX, IMP),
    (0x20, JSR, ABS), (0x21, AND, IZX), (0x22, XXX, IMP), (0x23, XXX, IMP), (0x24, BIT, ZP0), (0x25, AND, ZP0), (0x26, ROL, ZP0), (0x27, XXX, IMP),
    (0x30, BMI, REL), (0x31, AND, IZY), (0x32, XXX, IMP), (0x33, XXX, IMP), (0x34, NOP, IMP), (0x35, AND, ZPX), (0x36, ROL, ZPX), (0x37, XXX, IMP),
    (0x40, RTI, IMP), (0x41, EOR, IZX), (0x42, XXX, IMP), (0x43, XXX, IMP), (0x44, NOP, IMP), (0x45, EOR, ZP0), (0x46, LSR, ZP0), (0x47, XXX, IMP),
    (0x50, BVC, REL), (0x51, EOR, IZY), (0x52, XXX, IMP), (0x53, XXX, IMP), (0x54, NOP, IMP), (0x55, EOR, ZPX), (0x56, LSR, ZPX), (0x57, XXX, IMP),
    (0x60, RTS, IMP), (0x61, ADC, IZX), (0x62, XXX, IMP), (0x63, XXX, IMP), (0x64, NOP, IMP), (0x65, ADC, ZP0), (0x66, ROR, ZP0), (0x67, XXX, IMP),
    (0x70, BVS, REL), (0x71, ADC, IZY), (0x72, XXX, IMP), (0x73, XXX, IMP), (0x74, NOP, IMP), (0x75, ADC, ZPX), (0x76, ROR, ZPX), (0x77, XXX, IMP),
    (0x80, NOP, IMP), (0x81, STA, IZX), (0x82, NOP, IMP), (0x83, XXX, IMP), (0x84, STY, ZP0), (0x85, STA, ZP0), (0x86, STX, ZP0), (0x87, XXX, IMP),
    (0x90, BCC, REL), (0x91, STA, IZY), (0x92, XXX, IMP), (0x93, XXX, IMP), (0x94, STY, ZPX), (0x95, STA, ZPX), (0x96, STX, ZPY), (0x97, XXX, IMP),
    (0xA0, LDY, IMM), (0xA1, LDA, IZX), (0xA2, LDX, IMM), (0xA3, XXX, IMP), (0xA4, LDY, ZP0), (0xA5, LDA, ZP0), (0xA6, LDX, ZP0), (0xA7, XXX, IMP),
    (0xB0, BCS, REL), (0xB1, LDA, IZY), (0xB2, XXX, IMP), (0xB3, XXX, IMP), (0xB4, LDY, ZPX), (0xB5, LDA, ZPX), (0xB6, LDX, ZPY), (0xB7, XXX, IMP),
    (0xC0, CPY, IMM), (0xC1, CMP, IZX), (0xC2, NOP, IMP), (0xC3, XXX, IMP), (0xC4, CPY, ZP0), (0xC5, CMP, ZP0), (0xC6, DEC, ZP0), (0xC7, XXX, IMP),
    (0xD0, BNE, REL), (0xD1, CMP, IZY), (0xD2, XXX, IMP), (0xD3, XXX, IMP), (0xD4, NOP, IMP), (0xD5, CMP, ZPX), (0xD6, DEC, ZPX), (0xD7, XXX, IMP),
    (0xE0, CPX, IMM), (0xE1, SBC, IZX), (0xE2, NOP, IMP), (0xE3, XXX, IMP), (0xE4, CPX, ZP0), (0xE5, SBC, ZP0), (0xE6, INC, ZP0), (0xE7, XXX, IMP),
    (0xF0, BEQ, REL), (0xF1, SBC, IZY), (0xF2, XXX, IMP), (0xF3, XXX, IMP), (0xF4, NOP, IMP), (0xF5, SBC, ZPX), (0xF6, INC, ZPX), (0xF7, XXX, IMP),
}
```

Generated IR

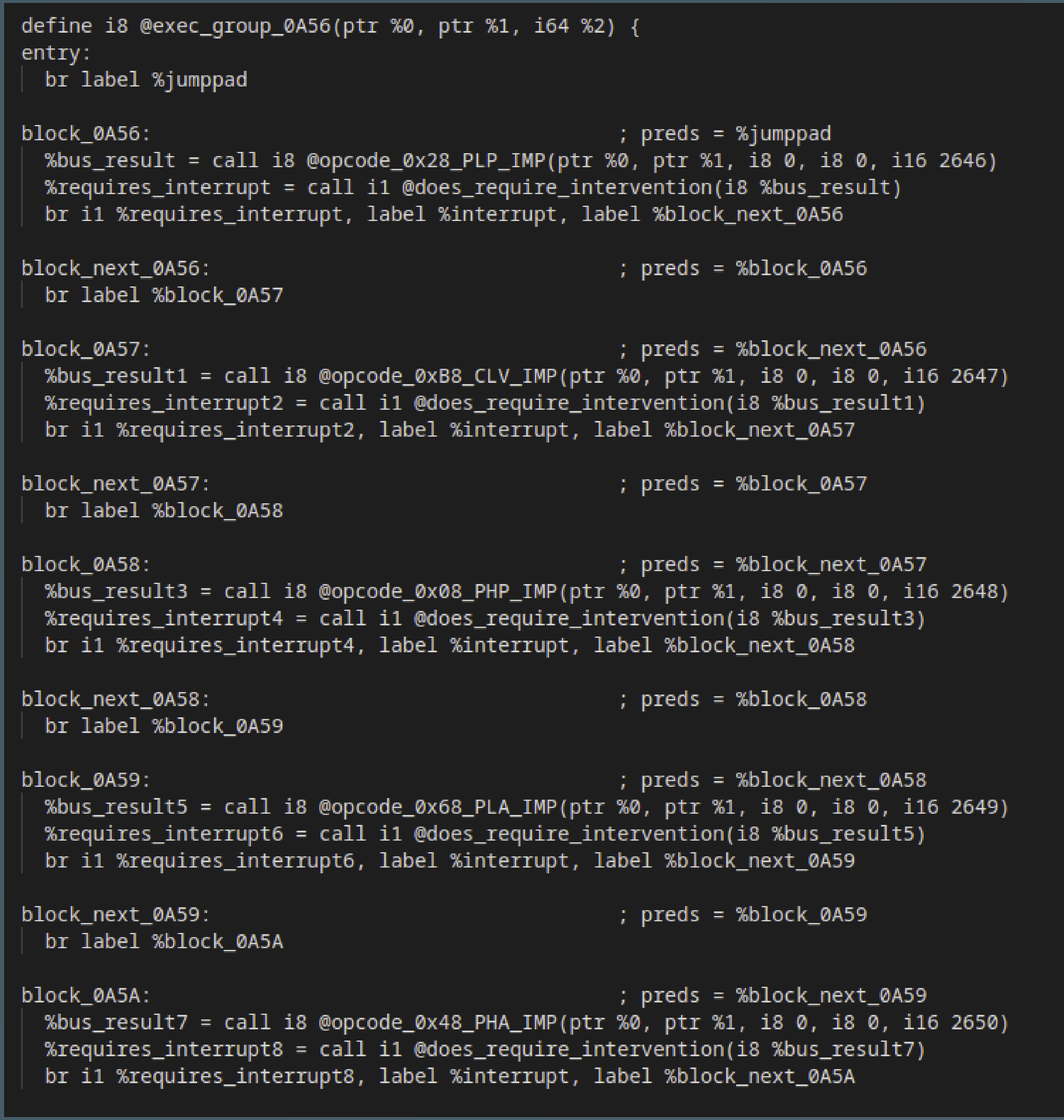
For 0xBC opcode, NOP instruction.
This one just increments the program counter.



```
; Function Attrs: alwaysinline mustprogress nofree norecurse nosync nounwind nonlazybind willreturn memory(argmem: readwrite) uwtable
define private noundef zeroext i8 @opcode_0x0C_NOP_IMP(ptr nocapture noundef %cpu, ptr nocapture noundef readnone %bus, i8 noundef zeroext %arg1,
start:
  %0 = getelementptr inbounds %"core_6502::core6502::Core6502", ptr %cpu, i64 0, i32 1
  %1 = load i16, ptr %0, align 8, !alias.scope !178, !noalias !181, !noundef !3
  %2 = add i16 %1, 1
  store i16 %2, ptr %0, align 8, !alias.scope !178, !noalias !181
  %3 = getelementptr inbounds %"core_6502::core6502::Core6502", ptr %cpu, i64 0, i32 6
  %bits.i = load i8, ptr %3, align 2, !alias.scope !178, !noalias !181, !noundef !3
  %bits6.i = or i8 %bits.i, 32
  %4 = load i64, ptr %cpu, align 8, !alias.scope !178, !noalias !181, !noundef !3
  %5 = add i64 %4, 4
  store i64 %5, ptr %cpu, align 8, !alias.scope !178, !noalias !181
  store i8 %bits6.i, ptr %3, align 2, !alias.scope !178, !noalias !181
  ret i8 0
}
```

Building the function

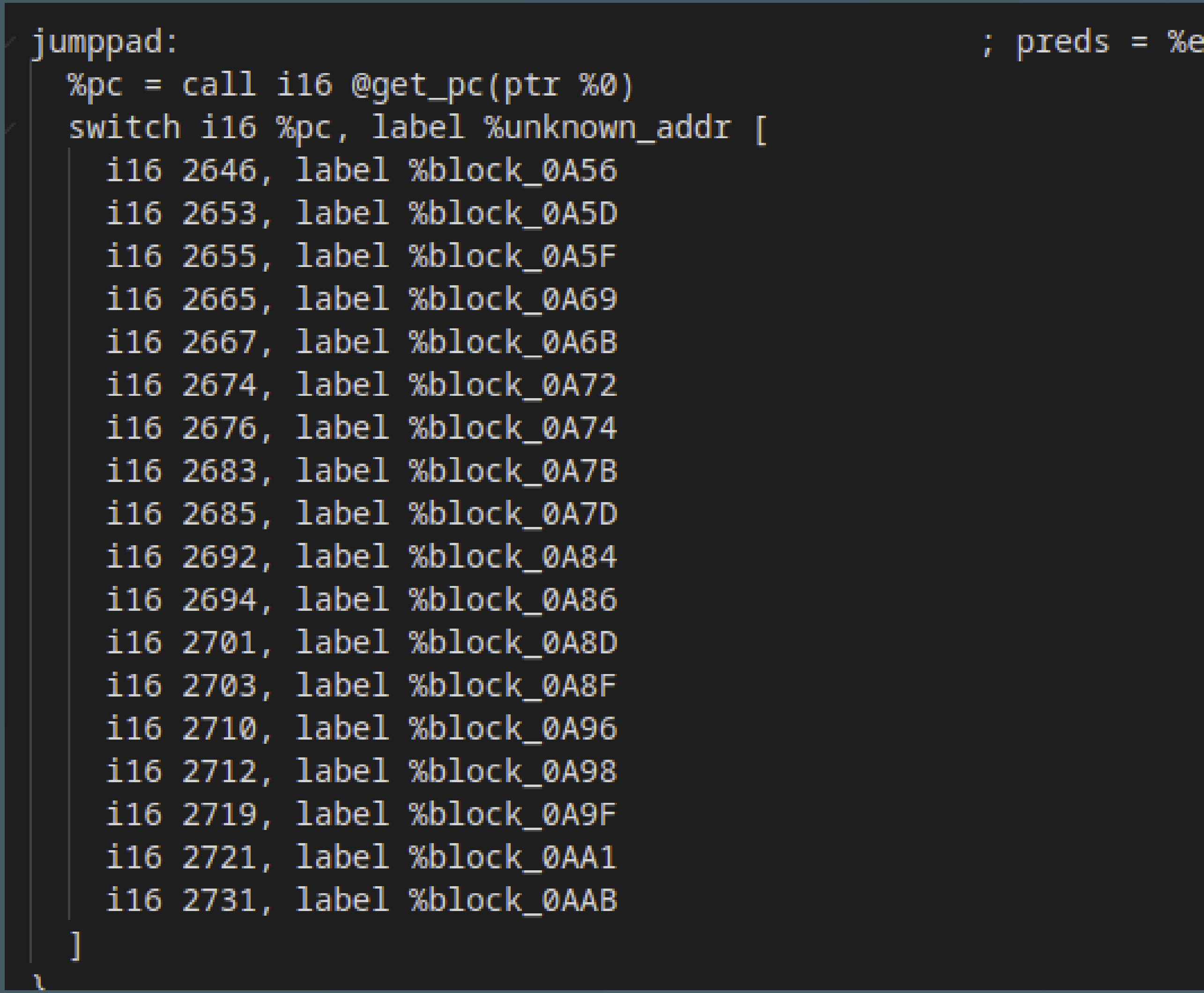
The pre-compiled functions get joined together as instructed by the 6502 bytecode.



```
define i8 @exec_group_0A56(ptr %0, ptr %1, i64 %2) {
entry:
  br label %jumppad

block_0A56:                                       ; preds = %jumppad
  %bus_result = call i8 @opcode_0x28_PLP_IMP(ptr %0, ptr %1, i8 0, i8 0, i16 2646)
  %requires_interrupt = call i1 @does_require_intervention(i8 %bus_result)
  br i1 %requires_interrupt, label %interrupt, label %block_next_0A56

block_next_0A56:                                  ; preds = %block_0A56
  br label %block_0A57

block_0A57:                                       ; preds = %block_next_0A56
  %bus_result1 = call i8 @opcode_0xB8_CLV_IMP(ptr %0, ptr %1, i8 0, i8 0, i16 2647)
  %requires_interrupt2 = call i1 @does_require_intervention(i8 %bus_result1)
  br i1 %requires_interrupt2, label %interrupt, label %block_next_0A57

block_next_0A57:                                  ; preds = %block_0A57
  br label %block_0A58

block_0A58:                                       ; preds = %block_next_0A57
  %bus_result3 = call i8 @opcode_0x08_PHP_IMP(ptr %0, ptr %1, i8 0, i8 0, i16 2648)
  %requires_interrupt4 = call i1 @does_require_intervention(i8 %bus_result3)
  br i1 %requires_interrupt4, label %interrupt, label %block_next_0A58

block_next_0A58:                                  ; preds = %block_0A58
  br label %block_0A59

block_0A59:                                       ; preds = %block_next_0A58
  %bus_result5 = call i8 @opcode_0x68_PLA_IMP(ptr %0, ptr %1, i8 0, i8 0, i16 2649)
  %requires_interrupt6 = call i1 @does_require_intervention(i8 %bus_result5)
  br i1 %requires_interrupt6, label %interrupt, label %block_next_0A59

block_next_0A59:                                  ; preds = %block_0A59
  br label %block_0A5A

block_0A5A:                                       ; preds = %block_next_0A59
  %bus_result7 = call i8 @opcode_0x48_PHA_IMP(ptr %0, ptr %1, i8 0, i8 0, i16 2650)
  %requires_interrupt8 = call i1 @does_require_intervention(i8 %bus_result7)
  br i1 %requires_interrupt8, label %interrupt, label %block_next_0A5A
```

The Jumppad

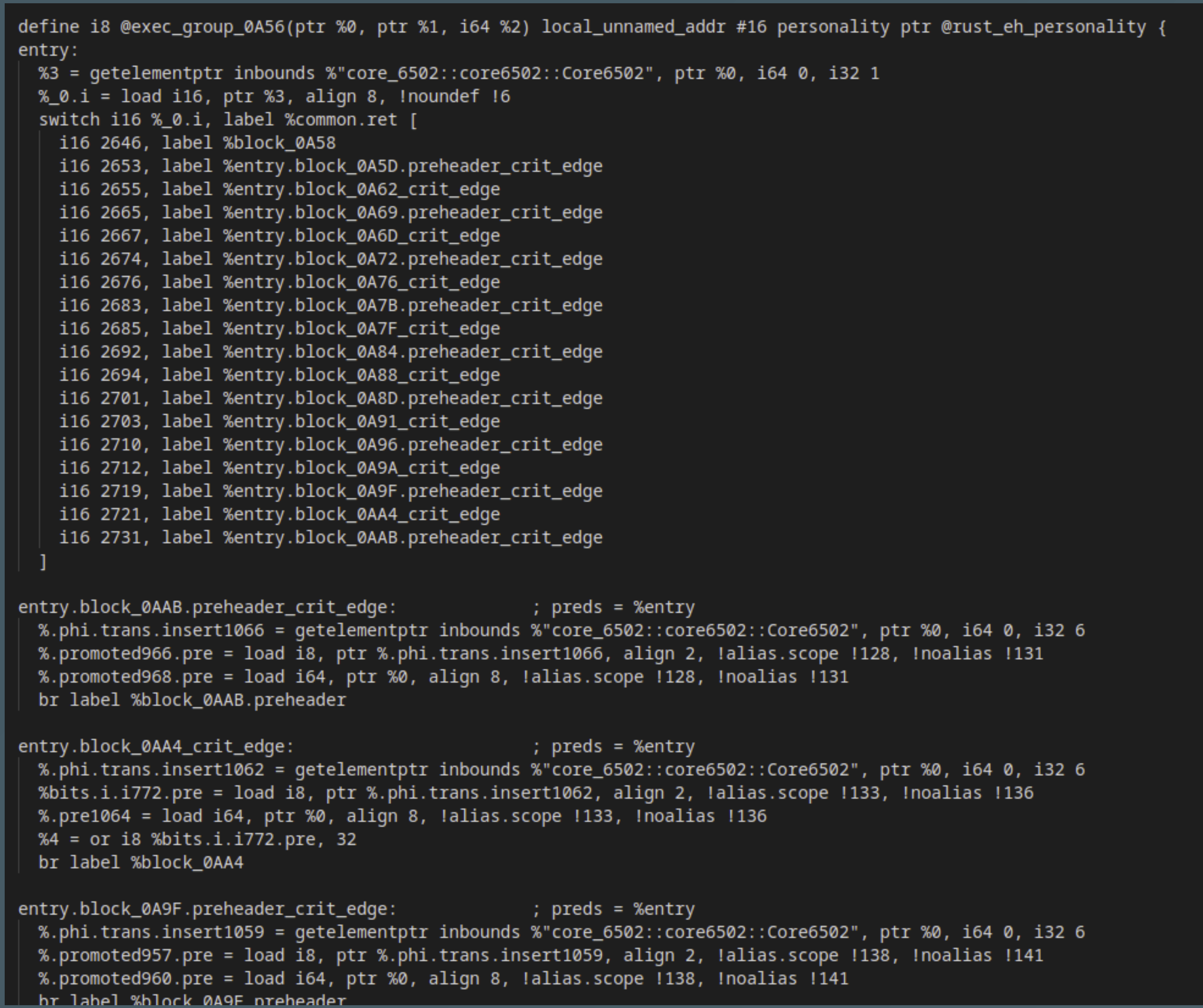
A block responsible for resuming a JIT'd function based on the current emulator program counter.



```
jumppad:                                             ; preds = %e
  %pc = call i16 @get_pc(ptr %0)
  switch i16 %pc, label %unknown_addr [
    i16 2646, label %block_0A56
    i16 2653, label %block_0A5D
    i16 2655, label %block_0A5F
    i16 2665, label %block_0A69
    i16 2667, label %block_0A6B
    i16 2674, label %block_0A72
    i16 2676, label %block_0A74
    i16 2683, label %block_0A7B
    i16 2685, label %block_0A7D
    i16 2692, label %block_0A84
    i16 2694, label %block_0A86
    i16 2701, label %block_0A8D
    i16 2703, label %block_0A8F
    i16 2710, label %block_0A96
    i16 2712, label %block_0A98
    i16 2719, label %block_0A9F
    i16 2721, label %block_0AA1
    i16 2731, label %block_0AAB
  ]
```

Optimized instructions

It's scary what kinds of optimizations LLVM comes up with.



```
define i8 @exec_group_0A56(ptr %0, ptr %1, i64 %2) local_unnamed_addr #16 personality ptr @rust_eh_personality {
entry:
  %3 = getelementptr inbounds %"core_6502::core6502::Core6502", ptr %0, i64 0, i32 1
  %_0.i = load i16, ptr %3, align 8, !noundef !6
  switch i16 %_0.i, label %common.ret [
    i16 2646, label %block_0A58
    i16 2653, label %entry.block_0A5D.preheader_crit_edge
    i16 2655, label %entry.block_0A62_crit_edge
    i16 2665, label %entry.block_0A69.preheader_crit_edge
    i16 2667, label %entry.block_0A6D_crit_edge
    i16 2674, label %entry.block_0A72.preheader_crit_edge
    i16 2676, label %entry.block_0A76_crit_edge
    i16 2683, label %entry.block_0A7B.preheader_crit_edge
    i16 2685, label %entry.block_0A7F_crit_edge
    i16 2692, label %entry.block_0A84.preheader_crit_edge
    i16 2694, label %entry.block_0A88_crit_edge
    i16 2701, label %entry.block_0A8D.preheader_crit_edge
    i16 2703, label %entry.block_0A91_crit_edge
    i16 2710, label %entry.block_0A96.preheader_crit_edge
    i16 2712, label %entry.block_0A9A_crit_edge
    i16 2719, label %entry.block_0A9F.preheader_crit_edge
    i16 2721, label %entry.block_0AA4_crit_edge
    i16 2731, label %entry.block_0AAB.preheader_crit_edge
  ]

entry.block_0AAB.preheader_crit_edge:             ; preds = %entry
  %.phi.trans.insert1066 = getelementptr inbounds %"core_6502::core6502::Core6502", ptr %0, i64 0, i32 6
  %.promoted966.pre = load i8, ptr %.phi.trans.insert1066, align 2, !alias.scope !128, !noalias !131
  %.promoted968.pre = load i64, ptr %0, align 8, !alias.scope !128, !noalias !131
  br label %block_0AAB.preheader

entry.block_0AA4_crit_edge:                       ; preds = %entry
  %.phi.trans.insert1062 = getelementptr inbounds %"core_6502::core6502::Core6502", ptr %0, i64 0, i32 6
  %bits.i.i772.pre = load i8, ptr %.phi.trans.insert1062, align 2, !alias.scope !133, !noalias !136
  %.pre1064 = load i64, ptr %0, align 8, !alias.scope !133, !noalias !136
  %4 = or i8 %bits.i.i772.pre, 32
  br label %block_0AA4

entry.block_0A9F.preheader_crit_edge:             ; preds = %entry
  %.phi.trans.insert1059 = getelementptr inbounds %"core_6502::core6502::Core6502", ptr %0, i64 0, i32 6
  %.promoted957.pre = load i8, ptr %.phi.trans.insert1059, align 2, !alias.scope !138, !noalias !141
  %.promoted960.pre = load i64, ptr %0, align 8, !alias.scope !138, !noalias !141
  br label %block_0A9F.preheader
```

No flashy demo I guess

But it passes a 6502 instruction accuracy test!

Thanks