undevs

Some crackhead students doing programming & Reverse-engineering

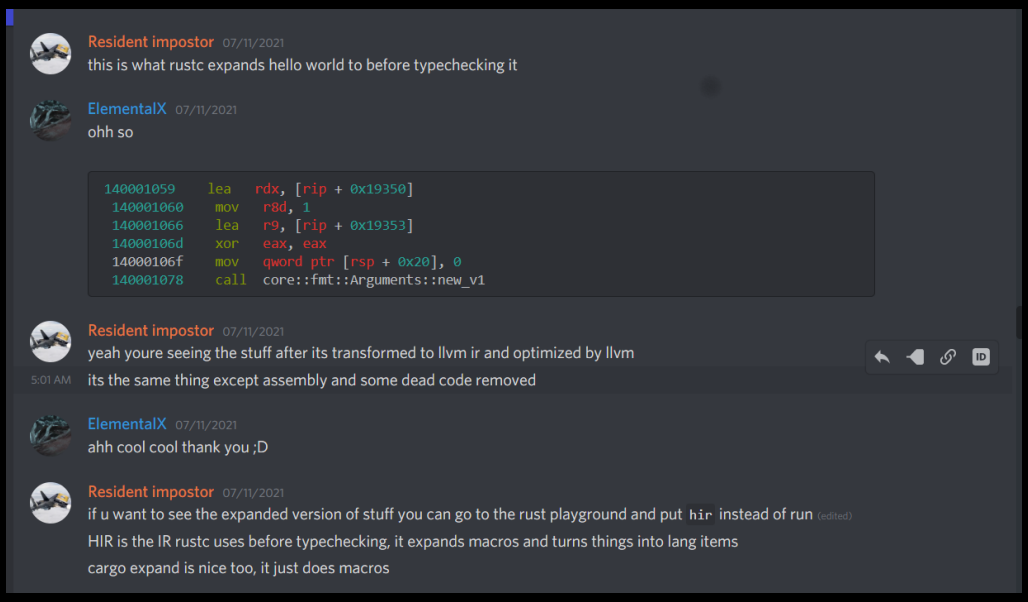
Twitter

Email

A Journey to understand LLVM-IR!

31 minute read

Why trying to understand LLVM-IR



So, a few days ago I saw some tweet saying about Reversing a Rust binary claiming it as a pretty tedious task, indeed it is to be honest, previously I tried to understand the disassembly of a rust binary but due to extremely irritating symbols in Rust, I would say I just gave up, but yes this time I was pretty determined to do it, so I wrote a small Rust program, and installed the cargo-disasm crate which helped me out to understand the disassembly of the exact functions without spitting out tons of weird symbol names, anyways I figured out later, but during this period I encountered macros in this small program, so someone at the Rust Programming language Discord introduced to me the term `LLVM IR` , till date I just knew there exists LLVM, so this *IR* ❓ was a new term for me. So I just became quite curious and just started to learn more how do I read an LLVM IR because it looks extremely weird.

How did I start the journey ❓

Just after a simple google search using the keyword understanding LLVM IR Code, I encounted this cool site known as freecompilercamp, which helped me to get to the know that LLVM IR is based on static single assignment representation, wait what the hell is a static single assignment representation? Let me explain this with a simple example:

```
// Source Language Representation : 1
a = c + d
b = a * k
a = b + i
b = d - n
a = a * b
f = c + d
```

The above example is a simple three add code, basically variables and operators, so now let us convert this into static single assignment representation, just remember we add a suffix to variables if we see they are being assigned and used multiple times.

```
//Target IR Representation : 2
a(1) = c(1) + d(1)
b(1) = a(1) * k
a(2) = b(1) + i
b(2) = d(1) - n
a(3) = a(2) + b(2)
f(1) = a(1)
```

Okay, so if we check out the above representation 1 the value inside **a(1)** & **f(1)** are same so we do not need any re-computation of that **f** variables, which somehow is a kind of optimization because we saved time to re-compute those variable and store them inside **f** , so this was just a small example of SSA-IR, LLVM-IR is based on this.

Taking up a very basic example using C

So, in the above section I just tried to make SSA representation quite simple, now it's time to write a simple C program and understand the overview of the LLVM-IR dump, I will follow the steps from the freecompilercamp and let's see what happens:

Step 1: Clang & LLVM Optimizer.

You need to check out if clang is installed on your machine, if not `sudo apt install clang` command will do the job for you, next check for the clang version `clang --version` hopefully your output will be something similar:

```
elemental@elemental-virtual-machine:~/Desktop$ clang --version
clang version 10.0.0-4ubuntu1
Target: x86_64-pc-linux-gnu
Thread model: posix
InstalledDir: /usr/bin
```

then next go ahead and check for LLVM's optimizer version `opt --version` , if it is not installed you can install this with the same command as above used to install clang.

Step 2: Writing a simple C program & generate the optimized LLVM-IR.

Once am done with checking the versions, the next step is to write a simple C program, compile it with Clang and then generate the optimized LLVM-IR

Tiny-C program:

```
#include <stdio.h>

int add(void){

int a = 5;
int b = 8;

int result = a + b;
return result;

}

int main(){

puts("Jump to Void");

add();

}
```

Compile it with it Clang the frontend of LLVM & emit un-optimized IR

```
clang -S -emit-llvm program.c
```

Optimize the LLVM-IR using opt

```
opt -S -mem2reg -instnamer program.ll
```

where, `-mem2reg` flag stands for move as many variables to registers as possible & `-instnamer` flag stands for assign names to anonymous instructions.

Now after, running this command I was left with a file named `program.ll` where the output is stored, the optimized output is as follows:

```
; ModuleID = 'program.c'
source_filename = "program.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

@.str = private unnamed_addr constant [13 x i8] c"Jump to Void\00", align 1
```

```
; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @add() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  store i32 5, i32* %1, align 4
  store i32 8, i32* %2, align 4
  %4 = load i32, i32* %1, align 4
  %5 = load i32, i32* %2, align 4
  %6 = add nsw i32 %4, %5
  store i32 %6, i32* %3, align 4
  %7 = load i32, i32* %3, align 4
  ret i32 %7
}

; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @main() #0 {
  %1 = call i32 @puts(i8* getelementptr inbounds ([13 x i8], [13 x i8]* @.str, i64 0, i64 0))
  %2 = call i32 @add()
  ret i32 0
}

declare dso_local i32 @puts(i8*) #1

attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-divide-sqrt-fp-math"="false" "disable-tail-calls"="false"
attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "frame-pointer"="all" "less-preci

!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{!"clang version 10.0.0-4ubuntu1 "}
```

Step 3:Understanding the above LLVM-IR

Now let us understand the above content, from

```
; ModuleID = 'program.c'
source_filename = "program.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"
```

Module ID is just a refernce to the current module then the next goes for source file name, the target data layout etc. The `e` specifies that the target laid out is in little-endian form, then `-m:e` states that llvm symbols are mangled and type of mangling is *ELF Mangling* here, then `-i64:64` states the alignment for an integer type of given bit, `f80:128` alignment of floating type in given bit, then `-n8:16:32:64` refers to the native integer widths for the target CPU in bits here it goes for *X86-64* then `S128` determines the natural alignment of the stack in bits. The last `target_triple` states about the *target-Vendor-OS*.

```
@.str = private unnamed_addr constant [13 x i8] c"Jump to Void\00", align 1

; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @add() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  store i32 5, i32* %1, align 4
  store i32 8, i32* %2, align 4
  %4 = load i32, i32* %1, align 4
  %5 = load i32, i32* %2, align 4
  %6 = add nsw i32 %4, %5
  store i32 %6, i32* %3, align 4
  %7 = load i32, i32* %3, align 4
  ret i32 %7
}
```

Now the `@.str = private unnamed_addr constant [13 x i8] c"Jump to Void\00", align 1` states that a string constant known as *.@str* is declared as a global constant, and the size of character is 8 bits so `[13 characters of 8 bit]` are initialized to that constant known as **"Jump to Void\00"** and `private` denotes that linkage are only accessible by objects in current module, `unnamed_address` denotes that the address of that string is exactly not known within the module, then `define dso_local i32 @add() #0` is the function definition for add `add()` , this `define` keyword is used while declaring functions in LLVM, `dso_local` specifies that the a function will resolve to a symbol within the same linkage unit, then `@add` is the name of function and `#0` is a function attribute, which denotes that the inliner should never inline this function in any **situation**. Then there are these temporary or local variables `%1` , `%2`, `%3`, `%4`, `%5`, `%6`, `%7`, which perform operations of allocating using `alloca` which reserves 4 a space of 4 bytes on the stack frame of the function `add()` then the other temporary variable do the same, then writing to memory is performed using `store` instruction the store instruction has two arguments a value which we want to store and a value to be stored in a certain memory address,here `5` is the value of type `i32` and is to be stored to a pointer of

type `i32` named `%1` . The next instruction at the next instruction of the integer pointer `%1` is now loaded to the variable named `%4` then the value of the integer pointer `%2` is loaded to the variable `%5` then both the values of 4 & 5 are now now added and stored inside variable `%6` now if we remember that variable we declared named `%3` but we never used, so now the value inside `%6` is stored inside the pointer `%3` , then the last variable `%7` the value stored in the pointer variable `*%3` is now stored to variable named `%7` and finally `%7` is returned.

Trying to re-write this a bit similar to the original source code.

Remember, the SSA IR? Yes, we are going to use the same suffix notation here so don't get confused with the terms.

```
define dso_local i32 @add() #0 {

a(1) = alloca i32, align 4
b(1) = alloca i32, allign 4
a(2) = alloca i32, allign 4
store i32 5, i32 *a(1) , allign 4
store i32 6, i32 *b(1), align 4
b(2) = load i32, i32 *a(1) , align 4
a(3) = load i32, i32 *b(1), align 4
a(4) = add nsw i32 b(2) + a(3)
store i32 a(4), i32 *a(2), align 4
result(1) = load i32, i32 *a(2), align 4
ret i32 result
}
```

I wrote this above code snippet to make it quite easy to relate to the original source code.

```
define dso_local i32 @main() #0 {
  %1 = call i32 @puts(i8* getelementptr inbounds ([13 x i8], [13 x i8]* @.str, i64 0, i64 0))
  %2 = call i32 @add()
  ret i32 0
}

declare dso_local i32 @puts(i8*) #1
```

Now after understanding the add() function, we are at the main() function where we have two variables `%1` & `%2` which are definitely temporary variables, in the first temporary variable w the `puts()` function which has a return type `i32` and which uses the getelementptr instruction and the semantics of it is `resultant variable = getelementptr inbounds <ty>, <ty>* <ptrval>{, [inrange] <ty> <idx>}*` then inside the next variable `%2` the call to add() function is stored and finally the `ret` type which is `i32` is present.

```
!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{!"clang version 10.0.0-4ubuntu1 "}
```

Then the last part has module flags contains a list of metadata triplets to communicate information about the module as a whole, then finally `!1 = !{!"clang version 10.0.0-4ubuntu1 "}` contains the compiler information.

Finally, I learnt to understand LLVM-IR, not sure I can write it well, may be the next blog, who knows 😃 !Let's apply this on a different programming language known as Rust and see if we can understand without any discomfort.

Writing a basic Rust program and understand the LLVM-IR.

Let us write a small program in Rust similar to our previous C program:

```
fn add(){

    let a:i32 = 67;
    let b:i32 = 33;
    let c:i32 = a + b;
    println!("The value of a + b is : {}", c);
}

fn main(){

    println!("Jump to Void");
    add();
}
}
```

I would suggest you to use the Rust playground to generate the LLVM-IR , else if you want to generate the LLVM-IR in a Linux machine follow the following steps:

1. Save your rust code inside a file with extension `.rs` .
2. rustc programname.rs -emit=llvm-ir
3. opt -S -mem2reg -instnamer programname.ll
4. less main.ll

If you compile the above code you will land up with the following equivalent optimized IR:

```
; ModuleID = 'main.7rcbfp3g-cgu.0'
source_filename = "main.7rcbfp3g-cgu.0"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

%"core::fmt::Formatter" = type { [0 x i64], { i64, i64 }, [0 x i64], { i64, i64 }, [0 x i64], { {}*, [3 x i64]* }, [0 x i32], i3
%"core::fmt::::Opaque" = type {}
%"core::fmt::Arguments" = type { [0 x i64], { [0 x { [0 x i8]*, i64 }]*, i64 }, [0 x i64], { i64*, i64 }, [0 x i64], { [0 x { i8
%"unwind::libunwind::_Unwind_Exception" = type { [0 x i64], i64, [0 x i64], void (i32, %"unwind::libunwind::_Unwind_Exception"*)*
%"unwind::libunwind::_Unwind_Context" = type { [0 x i8] }

@vtable.0 = private unnamed_addr constant { void (i64**)*, i64, i64, i32 (i64**)*, i32 (i64**)*, i32 (i64**)* } { void (i64**)* (
@0 = private unnamed_addr constant <{ [8 x i8] }> <{ [8 x i8] c"d\00\00\00\00\00\00\00" }>, align 4
@alloc12 = private unnamed_addr constant <{ [24 x i8] }> <{ [24 x i8] c"The value of a + b is : " }>, align 1
@alloc14 = private unnamed_addr constant <{ [1 x i8] }> <{ [1 x i8] c"\0A" }>, align 1
@alloc13 = private unnamed_addr constant <{ i8*, [8 x i8], i8*, [8 x i8] }> <{ i8* getelementptr inbounds (<{ [24 x i8] }>, <{ [2
@1 = private unnamed_addr constant <{ i8*, [0 x i8] }> <{ i8* bitcast (<{ i8*, [8 x i8], i8*, [8 x i8] }>* @alloc13 to i8*), [0 x
@alloc1 = private unnamed_addr constant <{ [13 x i8] }> <{ [13 x i8] c"Jump to Void\0A" }>, align 1
@alloc2 = private unnamed_addr constant <{ i8*, [8 x i8] }> <{ i8* getelementptr inbounds (<{ [13 x i8] }>, <{ [13 x i8] }>* @all
@2 = private unnamed_addr constant <{ i8*, [0 x i8] }> <{ i8* bitcast (<{ i8*, [8 x i8] }>* @alloc2 to i8*), [0 x i8] zeroinitia
@alloc6 = private unnamed_addr constant <{ [0 x i8] }> zeroinitializer, align 8
@3 = private unnamed_addr constant <{ i8*, [0 x i8] }> <{ i8* getelementptr inbounds (<{ [0 x i8] }>, <{ [0 x i8] }>* @alloc6, i

; std::sys_common::backtrace::__rust_begin_short_backtrace
; Function Attrs: noinline nonlazybind uwtable
define internal void @_ZN3std10sys_common9backtrace28__rust_begin_short_backtrace17hbb8f669eb12dabd8E(void ()* nonnull %f) unname
start:
  %0 = alloca { i8*, i32 }, align 8
  %_5 = alloca {}, align 1
  %_3 = alloca {}, align 1
; call core::ops::function::FnOnce::call_once
  call void @_ZN4core3ops8function6FnOnce9call_once17ha00c993b29b4dbdbE(void ()* nonnull %f)
  br label %bb2

bb1:                                              ; preds = %bb4
  %1 = bitcast { i8*, i32 }* %0 to i8**
  %2 = load i8*, i8** %1, align 8
  %3 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %0, i32 0, i32 1
  %4 = load i32, i32* %3, align 8
  %5 = insertvalue { i8*, i32 } undef, i8* %2, 0
  %6 = insertvalue { i8*, i32 } %5, i32 %4, 1
  resume { i8*, i32 } %6

bb2:                                              ; preds = %start
; invoke core::hint::black_box
  invoke void @_ZN4core4hint9black_box17h689721f9005ec6c1E()
          to label %bb3 unwind label %cleanup

bb3:                                              ; preds = %bb2
  ret void

bb4:                                              ; preds = %cleanup
  br label %bb1

cleanup:                                          ; preds = %bb2
  %7 = landingpad { i8*, i32 }
          cleanup
  %8 = extractvalue { i8*, i32 } %7, 0
  %9 = extractvalue { i8*, i32 } %7, 1
  %10 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %0, i32 0, i32 0
  store i8* %8, i8** %10, align 8
  %11 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %0, i32 0, i32 1
  store i32 %9, i32* %11, align 8
  br label %bb4
}

; std::rt::lang_start
; Function Attrs: nonlazybind uwtable
define hidden i64 @_ZN3std2rt10lang_start17h57c50b2714710b43E(void ()* nonnull %main, i64 %argc, i8** %argv) unnamed_addr #1 {
start:
  %_7 = alloca i64*, align 8
  %0 = bitcast i64** %_7 to void ()**
  store void ()* %main, void ()** %0, align 8
  %_4.0 = bitcast i64** %_7 to {}*
; call std::rt::lang_start_internal
  %1 = call i64 @_ZN3std2rt19lang_start_internal17ha12a50f31e33d94fE({}* nonnull align 1 %_4.0, [3 x i64]* noalias readonly alig
  br label %bb1
```

```
bb1:                                              ; preds = %start
  ret i64 %1
}

; std::rt::lang_start::
; Function Attrs: nonlazybind uwtable
define internal i32 @"_ZN3std2rt10lang_start28_$u7b$$u7b$closure$u7d$$u7d$17h953dfa4360d945c6E"(i64** noalias readonly align 8 de
start:
  %0 = bitcast i64** %_1 to void ()**
  %_3 = load void ()*, void ()** %0, align 8, !nonnull !3
; call std::sys_common::backtrace::__rust_begin_short_backtrace
  call void @_ZN3std10sys_common9backtrace28__rust_begin_short_backtrace17hbb8f669eb12dabd8E(void ()* nonnull %_3)
  br label %bb1

bb1:                                              ; preds = %start
; call <() as std::process::Termination>::report
  %1 = call i32 @"_ZN54_$LT$$LP$$RP$$u20$as$u20$std..process..Termination$GT$6report17h6926bda5ad1af176E"()
  br label %bb2

bb2:                                              ; preds = %bb1
  ret i32 %1
}

; std::sys::unix::process::process_common::ExitCode::as_i32
; Function Attrs: inlinehint nonlazybind uwtable
define internal i32 @_ZN3std3sys4unix7process14process_common8ExitCode6as_i3217hf1b569cb017f6afcE(i8* noalias readonly align 1 de
start:
  %_2 = load i8, i8* %self, align 1
  %0 = zext i8 %_2 to i32
  ret i32 %0
}

; core::fmt::ArgumentV1::new
; Function Attrs: nonlazybind uwtable
define internal { i8*, i64* } @_ZN4core3fmt10ArgumentV13new17hc3abfe613a6d8431E(i32* noalias readonly align 4 dereferenceable(4)
start:
  %0 = alloca %"core::fmt::::Opaque"*, align 8
  %1 = alloca i1 (%"core::fmt::::Opaque"*, %"core::fmt::Formatter"*)*, align 8
  %2 = alloca { i8*, i64* }, align 8
  %3 = bitcast i1 (%"core::fmt::::Opaque"*, %"core::fmt::Formatter"*)** %1 to i1 (i32*, %"core::fmt::Formatter"*)**
  store i1 (i32*, %"core::fmt::Formatter"*)* %f, i1 (i32*, %"core::fmt::Formatter"*)** %3, align 8
  %_3 = load i1 (%"core::fmt::::Opaque"*, %"core::fmt::Formatter"*)*, i1 (%"core::fmt::::Opaque"*, %"core::fmt::Formatter"*)** %
  br label %bb1

bb1:                                              ; preds = %start
  %4 = bitcast %"core::fmt::::Opaque"** %0 to i32**
  store i32* %x, i32** %4, align 8
  %_5 = load %"core::fmt::::Opaque"*, %"core::fmt::::Opaque"** %0, align 8, !nonnull !3
  br label %bb2

bb2:                                              ; preds = %bb1
  %5 = bitcast { i8*, i64* }* %2 to %"core::fmt::::Opaque"**
  store %"core::fmt::::Opaque"* %_5, %"core::fmt::::Opaque"** %5, align 8
  %6 = getelementptr inbounds { i8*, i64* }, { i8*, i64* }* %2, i32 0, i32 1
  %7 = bitcast i64** %6 to i1 (%"core::fmt::::Opaque"*, %"core::fmt::Formatter"*)**
  store i1 (%"core::fmt::::Opaque"*, %"core::fmt::Formatter"*)* %_3, i1 (%"core::fmt::::Opaque"*, %"core::fmt::Formatter"*)** %7
  %8 = getelementptr inbounds { i8*, i64* }, { i8*, i64* }* %2, i32 0, i32 0
  %9 = load i8*, i8** %8, align 8, !nonnull !3
  %10 = getelementptr inbounds { i8*, i64* }, { i8*, i64* }* %2, i32 0, i32 1
  %11 = load i64*, i64** %10, align 8, !nonnull !3
  %12 = insertvalue { i8*, i64* } undef, i8* %9, 0
  %13 = insertvalue { i8*, i64* } %12, i64* %11, 1
  ret { i8*, i64* } %13
}

; core::fmt::Arguments::new_v1
; Function Attrs: inlinehint nonlazybind uwtable
define internal void @_ZN4core3fmt9Arguments6new_v117h05d0a45d0996b748E(%"core::fmt::Arguments"* noalias nocapture sret dereferen
start:
  %_4 = alloca { i64*, i64 }, align 8
  %1 = bitcast { i64*, i64 }* %_4 to {}**
  store {}* null, {}** %1, align 8
  %2 = bitcast %"core::fmt::Arguments"* %0 to { [0 x { [0 x i8]*, i64 }]*, i64 }*
  %3 = getelementptr inbounds { [0 x { [0 x i8]*, i64 }]*, i64 }, { [0 x { [0 x i8]*, i64 }]*, i64 }* %2, i32 0, i32 0
  store [0 x { [0 x i8]*, i64 }]* %pieces.0, [0 x { [0 x i8]*, i64 }]** %3, align 8
  %4 = getelementptr inbounds { [0 x { [0 x i8]*, i64 }]*, i64 }, { [0 x { [0 x i8]*, i64 }]*, i64 }* %2, i32 0, i32 1
  store i64 %pieces.1, i64* %4, align 8
  %5 = getelementptr inbounds %"core::fmt::Arguments", %"core::fmt::Arguments"* %0, i32 0, i32 3
  %6 = getelementptr inbounds { i64*, i64 }, { i64*, i64 }* %_4, i32 0, i32 0
  %7 = load i64*, i64** %6, align 8
  %8 = getelementptr inbounds { i64*, i64 }, { i64*, i64 }* %_4, i32 0, i32 1
  %9 = load i64, i64* %8, align 8
  %10 = getelementptr inbounds { i64*, i64 }, { i64*, i64 }* %5, i32 0, i32 0
  store i64* %7, i64** %10, align 8
  %11 = getelementptr inbounds { i64*, i64 }, { i64*, i64 }* %5, i32 0, i32 1
```

```
  store i64 %9, i64* %11, align 8
  %12 = getelementptr inbounds %"core::fmt::Arguments", %"core::fmt::Arguments"* %0, i32 0, i32 5
  %13 = getelementptr inbounds { [0 x { i8*, i64* }]*, i64 }, { [0 x { i8*, i64* }]*, i64 }* %12, i32 0, i32 0
  store [0 x { i8*, i64* }]* %args.0, [0 x { i8*, i64* }]** %13, align 8
  %14 = getelementptr inbounds { [0 x { i8*, i64* }]*, i64 }, { [0 x { i8*, i64* }]*, i64 }* %12, i32 0, i32 1
  store i64 %args.1, i64* %14, align 8
  ret void
}

; core::ops::function::FnOnce::call_once
; Function Attrs: nonlazybind uwtable
define internal i32 @"_ZN4core3ops8function6FnOnce40call_once$u7b$$u7b$vtable.shim$u7d$$u7d$17h1aeebd87e9e5e023E"(i64** %_1) unna
start:
  %_2 = alloca {}, align 1
  %0 = load i64*, i64** %_1, align 8, !nonnull !3
; call core::ops::function::FnOnce::call_once
  %1 = call i32 @_ZN4core3ops8function6FnOnce9call_once17h2847987bc1eeec69E(i64* nonnull %0)
  br label %bb1

bb1:                                              ; preds = %start
  ret i32 %1
}

; core::ops::function::FnOnce::call_once
; Function Attrs: nonlazybind uwtable
define internal i32 @_ZN4core3ops8function6FnOnce9call_once17h2847987bc1eeec69E(i64* nonnull %0) unnamed_addr #1 personality i32
start:
  %1 = alloca { i8*, i32 }, align 8
  %_2 = alloca {}, align 1
  %_1 = alloca i64*, align 8
  store i64* %0, i64** %_1, align 8
; invoke std::rt::lang_start::
  %2 = invoke i32 @"_ZN3std2rt10lang_start28_$u7b$$u7b$closure$u7d$$u7d$17h953dfa4360d945c6E"(i64** noalias readonly align 8 dere
          to label %bb1 unwind label %cleanup

bb1:                                              ; preds = %start
  br label %bb2

bb2:                                              ; preds = %bb1
  ret i32 %2

bb3:                                              ; preds = %cleanup
  br label %bb4

bb4:                                              ; preds = %bb3
  %3 = bitcast { i8*, i32 }* %1 to i8**
  %4 = load i8*, i8** %3, align 8
  %5 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %1, i32 0, i32 1
  %6 = load i32, i32* %5, align 8
  %7 = insertvalue { i8*, i32 } undef, i8* %4, 0
  %8 = insertvalue { i8*, i32 } %7, i32 %6, 1
  resume { i8*, i32 } %8

cleanup:                                          ; preds = %start
  %9 = landingpad { i8*, i32 }
          cleanup
  %10 = extractvalue { i8*, i32 } %9, 0
  %11 = extractvalue { i8*, i32 } %9, 1
  %12 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %1, i32 0, i32 0
  store i8* %10, i8** %12, align 8
  %13 = getelementptr inbounds { i8*, i32 }, { i8*, i32 }* %1, i32 0, i32 1
  store i32 %11, i32* %13, align 8
  br label %bb3
}

; core::ops::function::FnOnce::call_once
; Function Attrs: nonlazybind uwtable
define internal void @_ZN4core3ops8function6FnOnce9call_once17ha00c993b29b4dbdbE(void ()* nonnull %_1) unnamed_addr #1 {
start:
  %_2 = alloca {}, align 1
  call void %_1()
  br label %bb1

bb1:                                              ; preds = %start
  ret void
}

; core::ptr::drop_in_place
; Function Attrs: nonlazybind uwtable
define internal void @_ZN4core3ptr13drop_in_place17hee06a0696601e5f5E(i64** %_1) unnamed_addr #1 {
start:
  %0 = alloca {}, align 1
  ret void
}

; core::hint::black_box
```

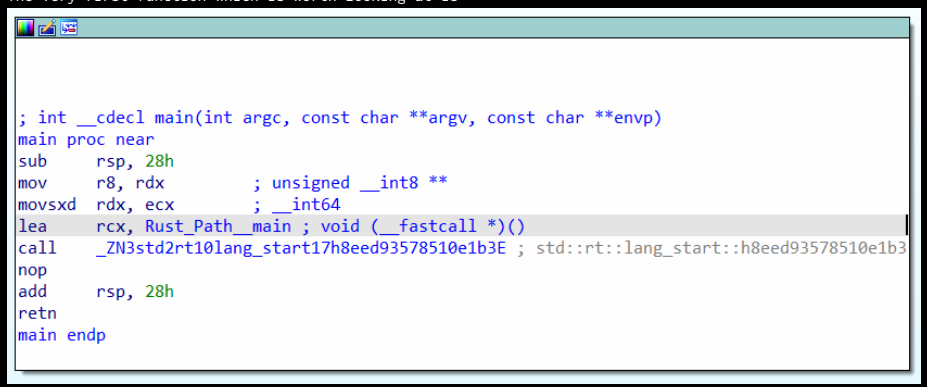
```
; Function Attrs: inlinehint nonlazybind uwtable
define internal void @_ZN4core4hint9black_box17h689721f9005ec6c1E() unnamed_addr #2 {
start:
  %dummy = alloca {}, align 1
  call void asm sideeffect "", "r,~{dirflag},~{fpsr},~{flags}"({}* %dummy), !srcloc !4
  ret void
}

; <() as std::process::Termination>::report
; Function Attrs: inlinehint nonlazybind uwtable
define internal i32 @"_ZN54_$LT$$LP$$RP$$u20$as$u20$std..process..Termination$GT$6report17h6926bda5ad1af176E"() unnamed_addr #2
start:
; call <std::process::ExitCode as std::process::Termination>::report
  %0 = call i32 @"_ZN68_$LT$std..process..ExitCode$u20$as$u20$std..process..Termination$GT$6report17h3aa1eed00d79f285E"(i8 0)
  br label %bb1

bb1:                                              ; preds = %start
  ret i32 %0
}

; <std::process::ExitCode as std::process::Termination>::report
; Function Attrs: inlinehint nonlazybind uwtable
define internal i32 @"_ZN68_$LT$std..process..ExitCode$u20$as$u20$std..process..Termination$GT$6report17h3aa1eed00d79f285E"(i8 %
start:
  %self = alloca i8, align 1
  store i8 %0, i8* %self, align 1
; call std::sys::unix::process::process_common::ExitCode::as_i32
  %1 = call i32 @_ZN3std3sys4unix7process14process_common8ExitCode6as_i3217hf1b569cb017f6afcE(i8* noalias readonly align 1 deref
  br label %bb1

bb1:                                              ; preds = %start
  ret i32 %1
}

; main::add
; Function Attrs: nonlazybind uwtable
define internal void @_ZN4main3add17hc2d1605dfece2271E() unnamed_addr #1 {
start:
  %_14 = alloca i32*, align 8
  %_13 = alloca [1 x { i8*, i64* }], align 8
  %_6 = alloca %"core::fmt::Arguments", align 8
  %c = alloca i32, align 4
  %_4.0 = load i32, i32* getelementptr inbounds ({ i32, i8 }, { i32, i8 }* bitcast (<{ [8 x i8] }>* @0 to { i32, i8 }*), i32 0, i
  %0 = load i8, i8* getelementptr inbounds ({ i32, i8 }, { i32, i8 }* bitcast (<{ [8 x i8] }>* @0 to { i32, i8 }*), i32 0, i32 1
  %_4.1 = trunc i8 %0 to i1
  store i32 %_4.0, i32* %c, align 4
  %_20 = load [2 x { [0 x i8]*, i64 }]*, [2 x { [0 x i8]*, i64 }]** bitcast (<{ i8*, [0 x i8] }>* @1 to [2 x { [0 x i8]*, i64 }]*
  %_7.0 = bitcast [2 x { [0 x i8]*, i64 }]* %_20 to [0 x { [0 x i8]*, i64 }]*
  store i32* %c, i32** %_14, align 8
  %arg0 = load i32*, i32** %_14, align 8, !nonnull !3
; call core::fmt::ArgumentV1::new
  %1 = call { i8*, i64* } @_ZN4core3fmt10ArgumentV13new17hc3abfe613a6d8431E(i32* noalias readonly align 4 dereferenceable(4) %arg
  %_17.0 = extractvalue { i8*, i64* } %1, 0
  %_17.1 = extractvalue { i8*, i64* } %1, 1
  br label %bb1

bb1:                                              ; preds = %start
  %2 = bitcast [1 x { i8*, i64* }]* %_13 to { i8*, i64* }*
  %3 = getelementptr inbounds { i8*, i64* }, { i8*, i64* }* %2, i32 0, i32 0
  store i8* %_17.0, i8** %3, align 8
  %4 = getelementptr inbounds { i8*, i64* }, { i8*, i64* }* %2, i32 0, i32 1
  store i64* %_17.1, i64** %4, align 8
  %_10.0 = bitcast [1 x { i8*, i64* }]* %_13 to [0 x { i8*, i64* }]*
; call core::fmt::Arguments::new_v1
  call void @_ZN4core3fmt9Arguments6new_v117h05d0a45d0996b748E(%"core::fmt::Arguments"* noalias nocapture sret dereferenceable(4
  br label %bb2

bb2:                                              ; preds = %bb1
; call std::io::stdio::_print
  call void @_ZN3std2io5stdio6_print17hd9977679df68edc4E(%"core::fmt::Arguments"* noalias nocapture dereferenceable(48) %_6)
  br label %bb3

bb3:                                              ; preds = %bb2
  ret void
}

; main::main
; Function Attrs: nonlazybind uwtable
define internal void @_ZN4main4main17h2d6c3d678af9e020E() unnamed_addr #1 {
start:
  %_2 = alloca %"core::fmt::Arguments", align 8
  %_11 = load [1 x { [0 x i8]*, i64 }]*, [1 x { [0 x i8]*, i64 }]** bitcast (<{ i8*, [0 x i8] }>* @2 to [1 x { [0 x i8]*, i64 }]*
  %_3.0 = bitcast [1 x { [0 x i8]*, i64 }]* %_11 to [0 x { [0 x i8]*, i64 }]*
  %_10 = load [0 x { i8*, i64* }]*, [0 x { i8*, i64* }]** bitcast (<{ i8*, [0 x i8] }>* @3 to [0 x { i8*, i64* }]**), align 8, !
; call core::fmt::Arguments::new_v1
  call void @_ZN4core3fmt9Arguments6new_v117h05d0a45d0996b748E(%"core::fmt::Arguments"* noalias nocapture sret dereferenceable(4
```

```
  br label %bb1

bb1:                                              ; preds = %start
; call std::io::stdio::_print
  call void @_ZN3std2io5stdio6_print17hd9977679df68edc4E(%"core::fmt::Arguments"* noalias nocapture dereferenceable(48) %_2)
  br label %bb2

bb2:                                              ; preds = %bb1
; call main::add
  call void @_ZN4main3add17hc2d1605dfece2271E()
  br label %bb3

bb3:                                              ; preds = %bb2
  ret void
}

; Function Attrs: nounwind nonlazybind uwtable
declare i32 @rust_eh_personality(i32, i32, i64, %"unwind::libunwind::_Unwind_Exception"*, %"unwind::libunwind::_Unwind_Context"*)

; std::rt::lang_start_internal
; Function Attrs: nonlazybind uwtable
declare i64 @_ZN3std2rt19lang_start_internal17ha12a50f31e33d94fE({}* nonnull align 1, [3 x i64]* noalias readonly align 8 dereferenceable(24), i64, i8** nocapture readonly, i8) unnamed_addr #1

; core::fmt::num::imp::<impl core::fmt::Display for i32>::fmt
; Function Attrs: nonlazybind uwtable
declare zeroext i1 @"_ZN4core3fmt3num3imp52_$LT$impl$u20$core..fmt..Display$u20$for$u20$i32$GT$3fmt17habdaec4fe3cabbf9E"(i32* noalias readonly align 4 dereferenceable(4), %"core::fmt::Formatter"* nonnull align 8 dereferenceable(64)) unnamed_addr #1

; std::io::stdio::_print
; Function Attrs: nonlazybind uwtable
declare void @_ZN3std2io5stdio6_print17hd9977679df68edc4E(%"core::fmt::Arguments"* noalias nocapture dereferenceable(48)) unnamed_addr #1

; Function Attrs: nonlazybind
define i32 @main(i32 %0, i8** %1) unnamed_addr #4 {
top:
  %2 = sext i32 %0 to i64
; call std::rt::lang_start
  %3 = call i64 @_ZN3std2rt10lang_start17h57c50b2714710b43E(void ()* @_ZN4main4main17h2d6c3d678af9e020E, i64 %2, i8** %1)
  %4 = trunc i64 %3 to i32
  ret i32 %4
}

attributes #0 = { noinline nonlazybind uwtable "probe-stack"="__rust_probestack" "target-cpu"="x86-64" }
attributes #1 = { nonlazybind uwtable "probe-stack"="__rust_probestack" "target-cpu"="x86-64" }
attributes #2 = { inlinehint nonlazybind uwtable "probe-stack"="__rust_probestack" "target-cpu"="x86-64" }
attributes #3 = { nounwind nonlazybind uwtable "probe-stack"="__rust_probestack" "target-cpu"="x86-64" }
attributes #4 = { nonlazybind "target-cpu"="x86-64" }

!llvm.module.flags = !{!0, !1, !2}

!0 = !{i32 7, !"PIC Level", i32 2}
!1 = !{i32 7, !"PIE Level", i32 2}
!2 = !{i32 2, !"RtLibUseGOT", i32 1}
!3 = !{}
!4 = !{i32 3109123}
!5 = !{i8 0, i8 2}
```

Well, fuck !

The equivalent LLVM-IR is way more large and bloated compared to that of C, but we have a lot of new functions here we will focus on only those important functions, I will be taking the help of a disassembler to understand which are the functions might be of our interest! I would suggest you to get a free copy of IDA-Disassembler.

The very first function which is worth looking at is



```
; int __cdecl main(int argc, const char **argv, const char **envp)
main proc near
sub     rsp, 28h
mov     r8, rdx         ; unsigned __int8 **
movsxd  rdx, ecx        ; __int64
lea     rcx, Rust_Path__main ; void (__fastcall *)()
call    _ZN3std2rt10lang_start17h8eed93578510e1b3E ; std::rt::lang_start::h8eed93578510e1b3
nop
add     rsp, 28h
retn
main endp
```

whose equivalent LLVM dump is :

```
; Function Attrs: nonlazybind
define i32 @main(i32 %0, i8** %1) unnamed_addr #4 {
top:
  %2 = sext i32 %0 to i64
; call std::rt::lang_start
  %3 = call i64 @_ZN3std2rt10lang_start17h57c50b2714710b43E(void ()* @_ZN4main4main17h2d6c3d678af9e020E, i64 %2, i8** %1)
  %4 = trunc i64 %3 to i32
  ret i32 %4
}
```

The above function `@main` with two arguments one which is `i32 %0` resembles to `int argc` and `*i8 ** %1` which resembles `char **argv` which is at an unnamed address at this modules with attribute `#4` then we have the `top:` block being declared now using `sext` instruction the value is type casted from `value1` to `value2` where both are integers and the bit of the value1 must be smaller here it is i32 than destination value2 and here it is i64, then another function named `@_ZN3std2rt10lang_start17h57c50b2714710b43E` is called which is of type `i64` and has three arguments one of them is a pointer to `@_ZN4main4main17h2d6c3d678af9e020E` function and other arguments are `i64 %2` which is `argc` & `i8** %1` which is `(unsigned _int8 **)argv)` , and this is stored in temporary variable `%3` then using the `trunc` instruction the value in temporary variable `%3` is truncated to `i32` from `i64` , and then finally the entire value stored in `%4` after the trunc instruction is returned.

Next function, is the ZN3std2rt10lang_start17h57c50b2714710b43E

```
; __int64 __fastcall std::rt::lang_start::h8eed93578510e1b3(void (__fastcall *)(), __int64, unsigned __int8 **)
_ZN3std2rt10lang_start17h8eed93578510e1b3E proc near

var_28= qword ptr -28h
var_20= qword ptr -20h
var_18= qword ptr -18h
var_10= qword ptr -10h
var_8= qword ptr -8

sub     rsp, 48h
mov     r9, r8
mov     r8, rdx
mov     [rsp+48h+var_18], rcx
mov     [rsp+48h+var_10], r8
mov     [rsp+48h+var_8], r9
mov     [rsp+48h+var_20], rcx
lea     rcx, [rsp+48h+var_20]
lea     rdx, vtable
call    _ZN3std2rt19lang_start_internal17h053a73b6001d3101E ; std::rt::lang_start_internal::h053a73b6001d3101
mov     [rsp+48h+var_28], rax
mov     rax, [rsp+48h+var_28]
add     rsp, 48h
retn
_ZN3std2rt10lang_start17h8eed93578510e1b3E endp
```

function, although it's not quite of our use, I am dumping the equivalent LLVM-IR of it:

```
define hidden i64 @_ZN3std2rt10lang_start17h57c50b2714710b43E(void ()* nonnull %main, i64 %argc, i8** %argv) unnamed_addr #1 {
start:
  %_7 = alloca i64*, align 8
  %0 = bitcast i64** %_7 to void ()**
  store void ()* %main, void ()** %0, align 8
  %_4.0 = bitcast i64** %_7 to {}*
; call std::rt::lang_start_internal
  %1 = call i64 @_ZN3std2rt19lang_start_internal17ha12a50f31e33d94fE({}* nonnull align 1 %_4.0, [3 x i64]* noalias readonly alig
  br label %bb1

bb1:                                              ; preds = %start
  ret i64 %1
}
```

The next interesting function to look at is the `@_ZN4main4main17h2d6c3d678af9e020E` function which the LLVM-IR dump equivalent to our original `main` function of this program :

```
; void __fastcall Rust_Path::main()
Rust_Path__main proc near

var_38= qword ptr -38h
var_30= byte ptr -30h

sub     rsp, 58h
lea     rcx, [rsp+58h+var_30]
lea     rdx, off_14001A3F8 ; "Jump to Void\n"
mov     r8d, 1
lea     r9, vtable
xor     eax, eax
mov     [rsp+58h+var_38], 0
call    _ZN4core3fmt9Arguments6new_v117ha5db2fc0c7d864b6E ; core::fmt::Arguments::new_v1::ha5db2fc0c7d864b6
lea     rcx, [rsp+58h+var_30]
call    _ZN3std2io5stdio6_print17hf51581d068fed5c2E ; std::io::stdio::_print::hf51581d068fed5c2
call    Rust_Path__add
nop
add     rsp, 58h
retn
Rust_Path__main endp
```

```
; main::main
; Function Attrs: nonlazybind uwtable
define internal void @_ZN4main4main17h2d6c3d678af9e020E() unnamed_addr #1 {
start:
  %_2 = alloca %"core::fmt::Arguments", align 8
  %_11 = load [1 x { [0 x i8]*, i64 }]*, [1 x { [0 x i8]*, i64 }]** bitcast (<{ i8*, [0 x i8] }>* @2 to [1 x { [0 x i8]*, i64 }]
  %_3.0 = bitcast [1 x { [0 x i8]*, i64 }]* %_11 to [0 x { [0 x i8]*, i64 }]*
  %_10 = load [0 x { i8*, i64* }]*, [0 x { i8*, i64* }]** bitcast (<{ i8*, [0 x i8] }>* @3 to [0 x { i8*, i64* }]**), align 8, !
; call core::fmt::Arguments::new_v1
  call void @_ZN4core3fmt9Arguments6new_v117h05d0a45d0996b748E(%"core::fmt::Arguments"* noalias nocapture sret dereferenceable(4
  br label %bb1

bb1:                                              ; preds = %start
; call std::io::stdio::_print
  call void @_ZN3std2io5stdio6_print17hd9977679df68edc4E(%"core::fmt::Arguments"* noalias nocapture dereferenceable(48) %_2)
  br label %bb2

bb2:                                              ; preds = %bb1
; call main::add
  call void @_ZN4main3add17hc2d1605dfece2271E()
  br label %bb3

bb3:                                              ; preds = %bb2
  ret void
}
```

This above function is the original main function of this program, as usual there is a function definition using the `define` keyword, with return type `void` then there is the `start` label and bunch of temporary registers performing load operation, bitwise casting & then finally another it jumps to `@_ZN3std2io5stdio6_print17hd9977679df68edc4E` which prints the string “Jump to Void\00”, wait what’s `br label %bb1` here ? The `br` instruction is used to cause control flow to a different block which is `bb1` in the same function , and `br label %bb1` is an unconditional form of the branch which only takes label as it’s target. Then after reaching to `bb1` label it again jumps to `bb2` label and calls the function `@_ZN4main3add17hc2d1605dfece2271E()` , and finally jumps to the label `bb3` which has the terminator instruction `ret` . Next we will look into the last function of our interest `define internal void @_ZN4main3add17hc2d1605dfece2271E() unnamed_addr #1` .

Let’s dive into the add() function:

```
Rust_Path__add proc near

var_78= qword ptr -78h
var_70= qword ptr -70h
var_68= qword ptr -68h
var_5C= dword ptr -5Ch
var_58= byte ptr -58h
var_28= qword ptr -28h
var_20= qword ptr -20h
result= qword ptr -18h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_8= qword ptr -8

sub     rsp, 98h
mov     [rsp+98h+var_10], 43h ; 'C'
mov     [rsp+98h+var_C], 21h ; '!'
mov     eax, cs:dword_14001A3A0
mov     [rsp+98h+var_5C], eax
lea     rax, [rsp+98h+var_5C]
mov     [rsp+98h+result], rax
mov     rcx, [rsp+98h+result] ; result
mov     [rsp+98h+var_8], rcx
lea     rdx, _ZN4core3fmt3num3imp52_$LT$impl$u20$core__fmt__Display$u20$for$u20$i32$GT$3fmt17hcd40a4f578e7723cE ; core::result::Result (__fastcall *)(int *, core::fmt:
call    _ZN4core3fmt10ArgumentV13new17h795eae02fc9ea795E ; core::fmt::ArgumentV1::new::h795eae02fc9ea795
mov     [rsp+98h+var_70], rax
mov     [rsp+98h+var_68], rdx
```

```
lea     rax, [rsp+98h+var_5C]
mov     [rsp+98h+result], rax
mov     rcx, [rsp+98h+result] ; result
mov     [rsp+98h+var_8], rcx
lea     rdx, _ZN4core3fmt3num3imp52_$LT$impl$u20$core__fmt__Display$u20$for$u20$i32$GT$3fmt17hcd40a4f578e7723cE ; core::result::Result (__fastcall *)(int *, core::fmt:
call    _ZN4core3fmt10ArgumentV13new17h795eae02fc9ea795E ; core::fmt::ArgumentV1::new::h795eae02fc9ea795
mov     [rsp+98h+var_70], rax
mov     [rsp+98h+var_68], rdx
mov     rax, [rsp+98h+var_68]
mov     rcx, [rsp+98h+var_70]
mov     [rsp+98h+var_28], rcx
mov     [rsp+98h+var_20], rax
lea     r9, [rsp+98h+var_28]
lea     rcx, [rsp+98h+var_58]
lea     rdx, off_14001A3C8 ; "The value of a + b is : \n"
mov     r8d, 2
mov     [rsp+98h+var_78], 1
call    _ZN4core3fmt9Arguments6new_v117ha5db2fc0c7d864b6E ; core::fmt::Arguments::new_v1::ha5db2fc0c7d864b6
lea     rcx, [rsp+98h+var_58]
call    _ZN3std2io5stdio6_print17hf51581d068fed5c2E ; std::io::stdio::_print::hf51581d068fed5c2
nop
add     rsp, 98h
retn
Rust_Path__add endp
```

```
define internal void @_ZN4main3add17hc2d1605dfece2271E() unnamed_addr #1 {
start:
  %_14 = alloca i32*, align 8
  %_13 = alloca [1 x { i8*, i64* }], align 8
  %_6 = alloca %"core::fmt::Arguments", align 8
  %c = alloca i32, align 4
  %_4.0 = load i32, i32* getelementptr inbounds ({ i32, i8 }, { i32, i8 }* bitcast (<{ [8 x i8] }>* @0 to { i32, i8 }*), i32 0, i
  %0 = load i8, i8* getelementptr inbounds ({ i32, i8 }, { i32, i8 }* bitcast (<{ [8 x i8] }>* @0 to { i32, i8 }*), i32 0, i32 1
  %_4.1 = trunc i8 %0 to i1
  store i32 %_4.0, i32* %c, align 4
  %_20 = load [2 x { [0 x i8]*, i64 }]*, [2 x { [0 x i8]*, i64 }]** bitcast (<{ i8*, [0 x i8] }>* @1 to [2 x { [0 x i8]*, i64 }]*
  %_7.0 = bitcast [2 x { [0 x i8]*, i64 }]* %_20 to [0 x { [0 x i8]*, i64 }]*
  store i32* %c, i32** %_14, align 8
  %arg0 = load i32*, i32** %_14, align 8, !nonnull !3
; call core::fmt::ArgumentV1::new
  %1 = call { i8*, i64* } @_ZN4core3fmt10ArgumentV13new17hc3abfe613a6d8431E(i32* noalias readonly align 4 dereferenceable(4) %arg
  %_17.0 = extractvalue { i8*, i64* } %1, 0
  %_17.1 = extractvalue { i8*, i64* } %1, 1
  br label %bb1

bb1:                                              ; preds = %start
  %2 = bitcast [1 x { i8*, i64* }]* %_13 to { i8*, i64* }*
  %3 = getelementptr inbounds { i8*, i64* }, { i8*, i64* }* %2, i32 0, i32 0
  store i8* %_17.0, i8** %3, align 8
  %4 = getelementptr inbounds { i8*, i64* }, { i8*, i64* }* %2, i32 0, i32 1
  store i64* %_17.1, i64** %4, align 8
  %_10.0 = bitcast [1 x { i8*, i64* }]* %_13 to [0 x { i8*, i64* }]*
; call core::fmt::Arguments::new_v1
  call void @_ZN4core3fmt9Arguments6new_v117h05d0a45d0996b748E(%"core::fmt::Arguments"* noalias nocapture sret dereferenceable(48
  br label %bb2

bb2:                                              ; preds = %bb1
; call std::io::stdio::_print
  call void @_ZN3std2io5stdio6_print17hd9977679df68edc4E(%"core::fmt::Arguments"* noalias nocapture dereferenceable(48) %_6)
  br label %bb3

bb3:                                              ; preds = %bb2
  ret void
}
```

The above function as usual there is a function definition, then bunch of temporary variables are declared and load and store instructions are performed, then there is a jump instruction to the label `bb1` where values of the variables `a` & `b` are calculated then stored inside a temporary variable and then an unconditional jump to `bb2` where the print function is called and the result is printed and finally terminator instruction return is called and it ends.

Summary

So, this was my first encounter with LLVM-IR and just a random experiment to understand the dump, although I think I could only break down and simplify 65-70 % of the total dump, I look forward to improve my skills and learn more about this. Indeed it was fun learning a new thing. Shouts to `resident imposter` & xeroxz for helping me out and answering to my stupid questions. I look forward to explore these stuffs in future blogs. Please reach me out at discord **ElementalX#3463** , if you think there's things which needs to be fixed or wrong information is present, I look forward to improve myself. Good day ahead!

Tags: LLVM-IR

Categories: low-level-exploration

Updated: July 16, 2021

SHARE ON

Author : ElementalX

THIS CONTENT WAS SUBMITTED TO AXIAL