Minimalist WASM w/ Rust

This Talk Is...

- A look at the WebAssembly toolchain
- An overview of the Rust ecosystem around that toolchain
- A case-study on WebAssembly binary tuning for size
- Interesting (hopefully)

This Talk Is Not...

- A comprehensive study of size optimization techniques
- How LLVM Generates WebAssembly And Why It Does It A Certain Way™
- Practical (well, sort of, you'll see)

No Baggage. No Cargo.

Hello World is too complicated.

```
#![no_std]

#[no_mangle]
pub extern fn add(a: i32, b: i32) -> i32 {
      a + b
}
```

No Baggage. No Cargo.

```
rustc \
    --target=wasm32-unknown-unknown \  # wasm triple
    --emit llvm-ir \                   # we will use llvm tools for the further steps
    --crate-type staticlib \           # clump everything together
    -O \                               # release mode. get smol.
    -o add.ll \                        # output file
    add.rs
```

Panic Handler

```
error: `#[panic_handler]` function required, but not found
```

Will get stripped out by release mode but we must appease the compiler gods.

```
use core::panic::PanicInfo;

#[panic_handler]
fn panic(_info: &PanicInfo) -> ! {
    loop {}
}
```

LLVM Intermediate Representation

```
; ModuleID = 'add.3a1fbbbh-cgu.0'
source_filename = "add.3a1fbbbh-cgu.0"
target datalayout = "e-m:e-p:32:32-i64:64-n32:64-S128"
target triple = "wasm32-unknown-unknown"

; Function Attrs: norecurse nounwind readnone
define i32 @add(i32 %a, i32 %b) unnamed_addr #0 {
Start:
  %0 = add i32 %b, %a
  ret i32 %0
}

attributes #0 = { norecurse nounwind readnone "target-cpu"="generic" }
```

Compile Again. And Link.

```
llc \
    -march=wasm32 \                    # target wasm
    -filetype=obj \                    # output an object file
    -O3                                # maximum optimization level
    add.ll

wasm-ld \
    --no-entry \                       # no entry function
    --export=add \                     # just export the add symbol
    -zstack-size=$[8 * 1024 * 1024] \  # optionally set wasm memory size (ex: 8MiB)
    -o add.wasm \
    add.o
```

Debugging!

Web Assembly Binary Tools

```
wasm-objdump -x add.o                    # Pre-linking representation
wasm2wat -o add.wast add.wasm            # Post-linking representation
```

Run That

```
wasm.add(1, 3); // = 4
```

What If The Heap Existed

The World's 5 Minutest Allocator

```
extern { static __heap_base: usize; }
static mut BUMP_POINTER : isize = 0;

#[no_mangle]
unsafe extern fn malloc(n: isize) -> *const usize {
    let r : *const usize = (&__heap_base as *const usize).offset(BUMP_POINTER);
    BUMP_POINTER += n;
    r
}

#[no_mangle]
unsafe extern fn free(_p: *const usize) {
    // ohno.jpg
}
```

Slices & Sums

```
#[no_mangle]
pub extern fn sum(slice: &[i32]) -> i32 {
    slice.iter().sum()
}
```

Run That

```
const jsArray = [1, 2, 3];                          // data
const cArrayPointer = wasm.malloc(jsArray.length * 4);  // heap allocation
const cArray = new Uint32Array(                     // memory view
    Wasm.memory.buffer, cArrayPointer, jsArray.length
);
cArray.set(jsArray);                                // memcpy
wasm.sum(cArrayPointer, cArray.length);             // = 6
```

i sHoUlD r0lL mY oWn

wasm-bindgen is good

```
#![no_std]
extern crate wasm_bindgen;
use wasm_bindgen::prelude::wasm_bindgen;
use core::alloc::{GlobalAlloc, Layout};

struct BadAllocator;
unsafe impl GlobalAlloc for BadAllocator {...}
#[global_allocator]
static ALLOC: BadAllocator = BadAllocator;

#[wasm_bindgen]
pub fn add(a: i32, b: i32) -> i32 {
    a + b
}
#[wasm_bindgen]
pub fn sum(slice : &[i32]) -> i32 {
    slice.iter().sum()
}
```

wasm-pack is good.

```
[profile.release]
opt-level = "s"     # Oz is much, much worse for some reason
lto = true          # doesn't do anything but it's nice, you know?

[package.metadata.wasm-pack.profile.release]
wasm-opt = [
    '--strip-producers', # not great practise
    '-Oz',
]
```

Final hand rolled size: 205 bytes

Final wasm-pack size: 216 bytes

Differences

- Additional Exports in Hand Rolled
- Function ordering
- Additional instructions in Cargo version
  - Not gonna speculate here but come talk to me about my theories

Links

- https://github.com/TannerRogalsky/wasm-micro-rs
- https://github.com/WebAssembly/wabt
- https://github.com/rustwasm/wasm-pack
- https://github.com/rustwasm/wasm-bindgen
- https://github.com/WebAssembly/binaryen
- https://twitter.com/WuTangTan