Web Assembly: What does it do? Does it do things? Let's find out!

Andrew Zah <zah@andrewzah.com>

Rust Meetup Seoul

May 28, 2019

1. What is WASM?
2. Applications of WASM in Rust
3. A Real, Non-Trivial WASM App

Journey

Web Assembly's Definition

from webassembly.org (emphasis mine)

WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server applications.

What WASM is not

- WASM is not a programming language, though you can write by hand

What WASM is not

- WASM is not a programming language, though you can write by hand
- WASM isn't standalone: it needs a host

What WASM is not

- WASM is not a programming language, though you can write by hand
- WASM isn't standalone: it needs a host
- WASM is not intended to replace javascript

what is it actually, though?

- WASM's format isn't coupled to any OS or architecture

what is it actually, though?

- WASM's format isn't coupled to any OS or architecture
- Very similar to Java's bytecode or C#'s CLR

what is it actually, though?

- WASM's format isn't coupled to any OS or architecture
- Very similar to Java's bytecode or C#'s CLR
- Name and definition is a misnomer

what is it actually, though?

- WASM's format isn't coupled to any OS or architecture
- Very similar to Java's bytecode or C#'s CLR
- Name and definition is a misnomer
- It can run anywhere you can build a host

WASM's specification

- WASM is stack-based, not register-based

WASM's specification

- WASM is stack-based, not register-based
- WASM 1.0 only has 4 primitives

WASM's specification

- WASM is stack-based, not register-based
- WASM 1.0 only has 4 primitives
- 'i32', 'i64', 'f32', 'f64'

WASM's specification

- WASM is stack-based, not register-based
- WASM 1.0 only has 4 primitives
- 'i32', 'i64', 'f32', 'f64'
- No arrays

WASM's specification

- WASM is stack-based, not register-based
- WASM 1.0 only has 4 primitives
- 'i32', 'i64', 'f32', 'f64'
- No arrays
- no jmp instruction

memory

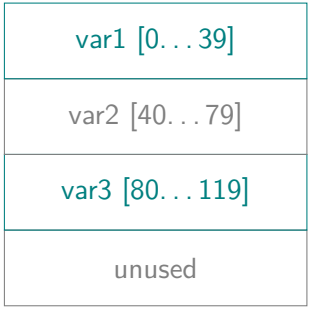
- WASM has no new::() operator or heap

memory

- WASM has no new::() operator or heap
- There are no objects or or garbage collection

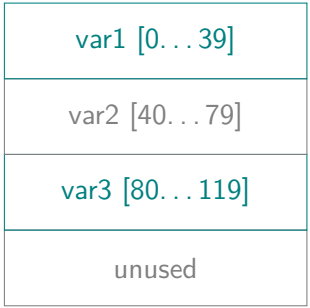
memory

- WASM has no new::() operator or heap
- There are no objects or or garbage collection
- Instead, WASM has linear memory



memory

- WASM has no new::() operator or heap
- There are no objects or or garbage collection
- Instead, WASM has linear memory
- this causes a need for linearization and bitmasking, etc



anyone can be a WASM host!

The Host needs to:

1. Load and validate the WASM binary (the file, not the logic)

anyone can be a WASM host!

The Host needs to:

1. Load and validate the WASM binary (the file, not the logic)
2. Expose Exports

anyone can be a WASM host!

The Host needs to:

1. Load and validate the WASM binary (the file, not the logic)
2. Expose Exports
3. Satisfy Imports

anyone can be a WASM host!

The Host needs to:

1. Load and validate the WASM binary (the file, not the logic)
2. Expose Exports
3. Satisfy Imports
4. Interpret & Execute Modules

anyone can be a WASM host!

The Host needs to:

1. Load and validate the WASM binary (the file, not the logic)
2. Expose Exports
3. Satisfy Imports
4. Interpret & Execute Modules
5. Isolate Modules

infinity and beyond?

- WebAssembly modules are hotswappable

infinity and beyond?

- WebAssembly modules are hotswappable
- PLCs (Programmable Logic Controllers) could adhere to a well-known contract
- This would let us program for hardware controllers in any language!

infinity and beyond?

- WebAssembly modules are hotswappable
- PLCs (Programmable Logic Controllers) could adhere to a well-known contract
- This would let us program for hardware controllers in any language!
- wasmi is an rust open-source wasm interpreter

infinity and beyond?

- WebAssembly modules are hotswappable
- PLCs (Programmable Logic Controllers) could adhere to a well-known contract
- This would let us program for hardware controllers in any language!
- wasmi is an rust open-source wasm interpreter
- serverless-wasm does indeed exist

security

- WASM binaries can always be converted back to plaintext

security

- WASM binaries can always be converted back to plaintext
- Do not put secrets in WASM modules

security

- WASM binaries can always be converted back to plaintext
- Do not put secrets in WASM modules
- Message handoffs are not always secure!

security

- WASM binaries can always be converted back to plaintext
- Do not put secrets in WASM modules
- Message handoffs are not always secure!
- Accordingly, sign and encrypt modules!

add example: rust side

main.rs

```
#[no_mangle]
pub extern "C" fn add_one(x: i32) -> i32 {
    x + 1
}
```

add example: generated WASM

main.wasm

```
(module
  (type $t0 (func (param i32) (result i32)))
  (func $add_one
      (export "add_one") (type $t0) (param $p0 i32) (result i32)
    get_local $p0
    i32.const 1
    i32.add)
  (table $T0 1 1 anyfunc)
  (memory $memory (export "memory") 17))
```

summary

- WASM is simply a instruction format that's portable
- WASM can run anywhere so long as a host exists
- Just because you can write WASM doesn't mean you should

Journey

wasm-bindgen

- CLI tool to generate bindings for the wasm code to work

wasm-bindgen

- CLI tool to generate bindings for the wasm code to work
- can be run manually but wasm-pack simplifies things

wasm-bindgen

- CLI tool to generate bindings for the wasm code to work
- can be run manually but wasm-pack simplifies things
- automates linearization, strings, complex types bindings

configuring cargo

Cargo.toml

```
[lib]
crate-type = ["cdylib"]

[build]
target = "wasm32-unknown-unknown"

[dependencies]
wasm-bindgen = "0.2.45"

[dependencies.rand]
features = ["wasm-bindgen"]
```

rust wasm: functions

main.rs

```
use wasm_bindgen::prelude::*;
#[wasm_bindgen]
extern "C" {
    #[wasm_bindgen(js_namespace = console)]
    fn log(s: &str);
    #[wasm_bindgen(method, structural, js_namespace = ROT)]
    fn draw(this: &Display, x: i32, y: i32, ch: &str);
}

#[wasm_bindgen]
pub fn greet(name: &str) {
    log(&format!("Hello, {}!", name));
}
```

js_sys and web_sys provide bindings

main.rs

```
use wasm_bindgen::prelude::*;
use wasm_bindgen::JsValue;
use web_sys::console::log_1;

#[wasm_bindgen]
pub fn hi() {
  log_1(JsValue::from_str("hi"));
}
```

rust wasm: structs and impls

main.rs

```
#[wasm_bindgen]
pub struct Counter { count: i32, }

#[wasm_bindgen]
impl Counter {
  pub fn get(&self) -> char {
    log(format!("Count: {}", self.count));
    self.count
  }
}
```

wasm-pack

- higher level wrapper for wasm-bindgen

wasm-pack

- higher level wrapper for wasm-bindgen
- commands: build, test, pack, publish, login

wasm-pack

- higher level wrapper for wasm-bindgen
- commands: build, test, pack, publish, login
- wasm-bindgen is still a dependency

wasm-pack output in /pkg

exa -T -L1 pkg/

```
pkg/
  package.json
  wasm_pack_test.d.ts
  wasm_pack_test.js
  wasm_pack_test_bg.d.ts
  wasm_pack_test_bg.wasm
```

wasm-pack output

main.js

```
const wasm = import('./pkg/hello_world');

wasm
  .then(m => m.greet('World!'))
  .catch(console.error);
```

extensions and alternatives

- web-sys and js-sys provide raw js/web API bindings

extensions and alternatives

- web-sys and js-sys provide raw js/web API bindings
- std-web has a similar goal but provides more JS interop + APIs

extensions and alternatives

- web-sys and js-sys provide raw js/web API bindings
- std-web has a similar goal but provides more JS interop + APIs
- rust-neon (external tool) generates Node.js modules

extensions and alternatives

- web-sys and js-sys provide raw js/web API bindings
- std-web has a similar goal but provides more JS interop + APIs
- rust-neon (external tool) generates Node.js modules
- yew is a web framework that compiles to WASM

optimizing the WASM output

- wasm-opt can significantly reduce binary size

optimizing the WASM output

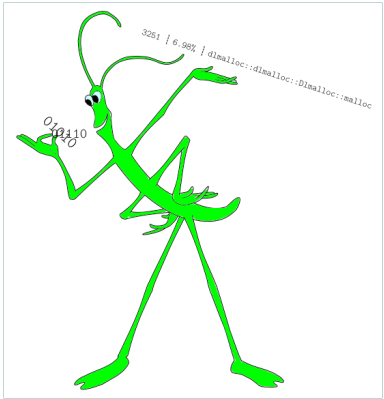
- wasm-opt can significantly reduce binary size
- wasm2wat can be used to confirm

optimizing the WASM output

- wasm-opt can significantly reduce binary size
- wasm2wat can be used to confirm
- twiggy is a wasm profiler that analyzes call graphs

optimizing the WASM output

- wasm-opt can significantly reduce binary size
- wasm2wat can be used to confirm
- twiggy is a wasm profiler that analyzes call graphs
- set lto = true in Cargo.toml, but limit to release only



Journey

korean apps

- Initially I wrote a cargo crate, korean-numbers

korean apps

- Initially I wrote a cargo crate, korean-numbers
- I wanted a way to use this logic in a webapp with React

korean apps

- Initially I wrote a cargo crate, korean-numbers
- I wanted a way to use this logic in a webapp with React
- I went with wasm-bindgen calls in package.json scripts

project structure

exa -T -L2 .

```
dist/
frontend/
native/
node_modules/
package.json
webpack.config.js
yarn.lock
```

why wasm-pack may be better

package.json

```
"scripts": {
  "build-debug": "cd native; cargo +nightly build
      --target wasm32-unknown-unknown
    && wasm-bindgen
       target/wasm32-unknown-unknown/debug/korean_wasm.wasm
      --out-dir ../frontend/wasm; cd .."
}
```

rust side: glue code

native/src/lib.rs

```
use korean_nums::{ NumberSystem, ...}
#[wasm_bindgen]
struct KoreanInteger { number: i128, hangeul: String, }

#[wasm_bindgen]
pub fn random_korean_int(
  lower_str: &str, upper_str: &str, num_system: &str)
  -> KoreanInteger
{ ... }
```

frontend layout

exa -T -L2 frontend/

```
components/
  korean_numbers/
index.html
index.js
wasm/
  korean_wasm.d.ts
  korean_wasm.js
  korean_wasm_bg.d.ts
  korean_wasm_bg.wasm
```

calling code from the frontend

frontend/components/korean_numbers/index.js

```
const { random_int } = require("../../wasm/korean_wasm");

const res = random_korean_int(0, 10, "sino");
```

non-trivial optimizations

ls -lah | rg wasm | awk 'print $5, $9' | sort

```
282K wasm-opt-Oz-flag.wasm
283K wasm-opt-Os-flag.wasm
306K wasm-opt-O3-flag.wasm
573K wasm-opt-default.wasm
695K bindgen-development.wasm
```

non-trivial optimizations

ls -lah | rg wasm | awk 'print $5, $9' | sort

```
282K wasm-opt-Oz-flag.wasm
283K wasm-opt-Os-flag.wasm
306K wasm-opt-O3-flag.wasm
573K wasm-opt-default.wasm
695K bindgen-development.wasm
```

release mode versions

```
126K wasm-opt-lto-Oz-flag.wasm
136K bindgen-release-lto.wasm
166K bindgen-release.wasm
```

inspecting with twiggy

twiggy top wasm-bindgen-default.wasm

```
Shallow Bytes  Shallow %  Item
124493 | 17.50% | "function names" subsection
57598  |  8.10% | rand_hc::hc128::Hc128Core::sixteen_steps
50394  |  7.09% | <rand_hc::hc128::Hc128Core>...::generate
24781  |  3.48% | data[1]
22978  |  3.23% | data[2]
 7624  |  1.07% | <rand::rngs::entropy::EntropyRng>...::try_fill_bytes
 6928  |  0.97% | korean_nums::parse::parse_hangeul_sino
 4918  |  0.69% | core::num::flt2dec::strategy::dragon::mul_pow10
```

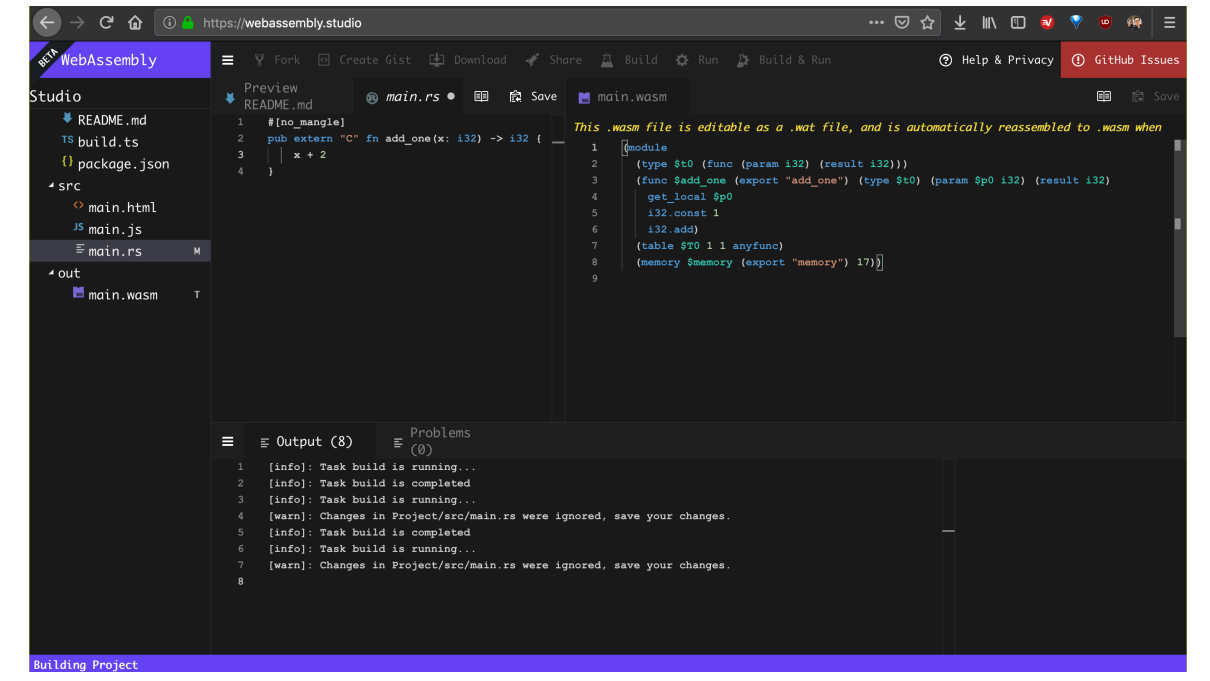
inspecting release mode with twiggy

twiggy top wasm-opt-Oz-release-lto.wasm

```
Shallow Bytes  Shallow %  Item
9753 | 7.55% | data[0]
8556 | 6.62% | code[33]
8457 | 6.54% | data[1]
7012 | 5.42% | code[76]
6156 | 4.76% | code[127]
6145 | 4.75% | code[75]
```

Thank you

Andrew Zah


https://webassembly.studio
BETA
WebAssembly
Studio
Fork
Create Gist
Download
Share
Build
Run
Build & Run
Help & Privacy
GitHub Issues
README.md
build.ts
package.json
src
main.html
main.js
main.rs
M
out
main.wasm
T
Preview README.md
main.rs
Save
#[no_mangle]
pub extern "C" fn add_one(x: i32) -> i32 {
x + 2
}
main.wasm
Save
This .wasm file is editable as a .wat file, and is automatically reassembled to .wasm when
(module
(type $t0 (func (param i32) (result i32)))
(func $add_one (export "add_one") (type $t0) (param $p0 i32) (result i32)
get_local $p0
i32.const 1
i32.add)
(table $T0 1 1 anyfunc)
(memory $memory (export "memory") 17))
Output (8)
Problems (0)
[info]: Task build is running...
[info]: Task build is completed
[info]: Task build is running...
[warn]: Changes in Project/src/main.rs were ignored, save your changes.
[info]: Task build is completed
[info]: Task build is running...
[warn]: Changes in Project/src/main.rs were ignored, save your changes.
Building Project

further reading

- WASM Homepage `https://webassembly.org`
- WASM Spec `https://webassembly.github.io/spec/`
- wasm-bindgen docs `https://rustwasm.github.io/docs/wasm-bindgen/`
- stdweb `https://github.com/koute/stdweb`
- twiggy profiler `https://github.com/rustwasm/twiggy`
- wat2wasm, wasm2wat, and more `https://github.com/WebAssembly/wabt`
- websys `https://rustwasm.github.io/wasm-bindgen/api/web_sys/`
- neon-bindings `https://github.com/neon-bindings/neon`
- optimizing emscriptem `https://emscripten.org/docs/optimizing/Optimizing-Code.html`
- wasmi: rust wasm interpreter `https://github.com/paritytech/wasmi`

bibliography

K. Hoffman.
Programming WebAssembly with Rust
Unified Development for Web, Mobile, and Embedded Applications
The Pragmatic Programmers, LLC
ISBN-13: 978-1-68050-636-5