moz://a

Using WebAssembly in *all* the Web

11/1/2018

Yury Delendik
Mozilla

Overview

- Introduction to WebAssembly
- How to create a WebAssembly code
- How to execute it
  - In web browsers
  - In node.js
  - Anywhere else?
- Why and when it is fast?

Assembly Language for the Web

- “JavaScript is Assembly Language for the Web”
- Now, WebAssembly is Assembly (Language) for the Web
- WebAssembly binary file is a container for
  - metadata: e.g. function signatures, memory size, exports, imports
  - machine code
  - data segments
  - custom information
- WebAssembly has text representation

Machine code

- Operators is matched to modern ISAs instructions
- Operands is expressed via stack (instead of registers)
- Locals, globals, memory
- Stack machine simplifies code verification

Simple program

```
(module
    (func (result i32)
        i32.const 42
    )
)
```

```
00000000: 00 61 73 6d 01 00 00 00 01 05 01 60 00 01 7f 03
00000010: 02 01 00 0a 06 01 04 00 41 2a 0b
```

Stack machine

$$\phi = \frac{1+\sqrt{5}}{2}$$

Operation	Stack
const 1	**1**
const 5	1 | **5**
sqrt	1 | **2.2360**
+	**3.2360**
const 2	3.2360 | **2**
/	**1.6180**

```
(func (export "gold") (result f64)
    f64.const 1
    f64.const 5
    f64.sqrt
    f64.add
    f64.const 2
    f64.div
)
```

Convert between binary and text formats

- There are tools
  - wabt
  - binaryen
- Web browsers devtools display text representation

```
$ # wabt
$ wat2wasm gold.wat -o gold.wasm
$ wasm2wat gold.wasm -o gold.wat
```

Load and execute WebAssembly

- Web Browsers (and JS engines) implement JS API for WebAssembly
  - WebAssembly object contains
    - methods to load and instantiate module
    - misc. utils to inspect module

```
const buffer = await (await fetch('gold.wasm')).arrayBuffer();
const { instance } = await WebAssembly.instantiate(buffer);
// or
const request = fetch('gold.wasm'); // has to be 'application/wasm'
const { instance } = await WebAssembly.instantiateStreaming(request);

console.log(instance.exports.gold());
```

Load and execute WebAssembly in e.g. node.js

- node.js is based on v8
- v8 implements WebAssembly
- Can node.js run my wasm file?

```
// node --experimental-repl-await
const buffer = require('fs').readFileSync('gold.wasm');
const { instance } = await WebAssembly.instantiate(buffer);
instance.exports.gold();
```

Importing the world

```
(module
  (import "Math" "exp" (func $exp (param f64) (result f64)))
  (func (export "sigmoid") (param $x f64) (result f64)
     f64.const 1
     f64.const 1
     get_local $x
     f64.neg
     call $exp
     f64.add
     f64.div))
```

$$S(x) = \frac{1}{1+e^{-x}}$$

Sigmoid Formatted

```
(module
 (import "Math" "exp" (func $exp (param f64) (result f64)))
 (export "sigmoid" (func $sigmoid))
 (func $sigmoid (param $x f64) (result f64)
  (f64.div
     (f64.const 1)
     (f64.add
       (f64.const 1)
       (call $exp (f64.neg  (get_local $x)))
     )
  ) ))
```

World to import

- WebAssembly.instantiate allows to specify required imports
- Constants, memory, tables, functions can be imported.

```
const buffer = await (await fetch('sigmoid.wasm')).arrayBuffer();
const imports = { Math: { exp: Math.exp } };
const { instance } = await WebAssembly.instantiate(buffer, imports);

console.log(instance.exports.sigmoid(0));
```

Is WebAssembly fast?

- Produced native code is efficient
- Calls from/to JS can be inefficient
- Marshalling is expensive

Demo

- FFT
- Rust

More tools

- Compile C code to WebAssembly
  - emscripten
  - LLVM (no libc, libc++ yet) -- wasm32-unknown-unknown-wasm
- Compile Rust code to WebAssembly
  - Supports wasm32-unknown-unknown target
  - wasm-bindgen / wasm-pack / js-sys / web-sys
  - https://rustwasm.github.io/book/
- WebAssembly Studio [https://webassembly.studio/]
- Did you try pyodide [https://iodide.io/pyodide-demo/python.html]?

Can WebAssembly live without JavaScript?

- Cranelift - A native code generator
- WAVM - WebAssembly Virtual Machine
- wasmjit - Kernel Mode WebAssembly Runtime for Linux
- Nebulet - A microkernel that implements a WebAssembly “usermode”
- wasmi - WebAssembly interpreter

Thank You