Part 1: Introduction to WebAssembly
What is WebAssembly?

Bringing the Web up to Speed with WebAssembly

Andreas Haas  Andreas Rossberg  Derek L. Schuff†  Ben L. Titzer
Google GmbH, Germany / "Google Inc, USA
{ahaas,rossberg,dschuff,titzer}@google.com

Dan Gohman  Luke Wagner  Alon Zakai
Mozilla Inc, USA
{sunfishcode,luke,azakai}@mozilla.com

Michael Holman
Microsoft Inc, USA
michael.holman@microsoft.com

JF Bastien
Apple Inc, USA
jfbastien@apple.com

Abstract
The maturation of the Web platform has given rise to sophisticated and demanding Web applications such as interactive 3D visualization, audio and video software, and games. With that, efficiency and security of code on the Web has become more important than ever. Yet JavaScript as the only built-in language of the Web is not well-equipped to meet these device types. By historical accident, JavaScript is the only natively supported programming language on the Web, its widespread usage unmatched by other technologies available only via plugins like ActiveX, Java or Flash. Because of JavaScript’s ubiquity, rapid performance improvements in modern VMs, and perhaps through sheer necessity, it has become a compilation target for other languages. Through Emscripten 1.431, even C and C++ programs can be compiled to

Haas et al., PLDI 2017

https://webassembly.org/
What is WebAssembly?

● “Byte code for the web”
  ○ Very rough idea: “JVM, without 3rd party plugins, directly in the browser”
  ○ Lower-level than JavaScript

● Fast
  ○ Compact binary format → quicker to parse
  ○ Instructions map closely to common hardware
  ○ Linear memory, no garbage collector

● Safe
  ○ Typed, separated code and data, modules, ...

● Portable
  ○ Support by all 4 major browsers, x86/ARM
Use Cases

- Designed as compilation target
  - For low-level languages
  - From C, C++ with Emscripten
  - Rust, Go, ...

- Application domains
  - Audio/video processing, codecs, e.g., AV1
  - Compression, e.g., Brotli
  - Games, simulations, e.g., Unreal Engine 4
  - Language runtimes, e.g., Blazor (C#)
  - Complex web applications, e.g., AutoCAD

Binary and Text Format

https://webassembly.studio/
Text Format: Overview

- Syntax: S-expressions
- 1 file = 1 module
- 1 module : n sections
  - Types
  - Imports/Exports
  - Functions
  - Globals
  - Data

```
(module
  (type $t0 (func (param i32 i32) (result i32)))
  ...
  (import "env" "b" (func $env.b (type $t4)))
  ...
  (func $f7 (param i32 i32) (result i32)
    (local $l0 i32) ...
    get_global $g2
    i32.const 48
    i32.add
    set_global $g2
    ...
    i32.load
    ...
  (global $g2 (mut i32))
  (export "h" (func $h))
  (data (i32.const 1152) "Hello, world!")
)
```
• Code is organized in functions
  ○ No OOP: no objects or methods
• Globals, per-function locals
• Stack machine
• 4 types: i32, i64, f32, f64
• Linear, unmanaged memory
  ○ 32-bit addresses

---

```
(module
  (type $t0 (func (param i32 i32) (result i32)))
...
  (import "env" "b" (func $env.b (type $t4)))
...
  (func $f7 (param i32 i32) (result i32)
    (local $l0 i32) ...
    get_global $g2
    i32.const 48
    i32.add
    set_global $g2
    ...
    i32.load
    ...
  (global $g2 (mut i32))
  (export "h" (func $h))
  (data (i32.const 1152) "Hello, world!")
)
```
Structured Control-Flow

- Well-nested blocks
  - Branches (br) reference blocks by numerical label

- Block types: block, if, loop

```plaintext
block
  ...
  br_if 0
  ...
end

...  ;; infinite loop
loop
  ...
  br 0
  ...
end

...  i32.const 0
if
  ...
else
  ...
end

...```
Host Embedding

- WebAssembly itself is agnostic of its *host*
  - No functions defined by default
  - No system calls/IO routines by default

- Most common embedding: web browsers
WebAssembly API in JavaScript

```javascript
let importObject = {
    env: {
        print: function(s) {
            console.log(s);
        },
    },
};

fetch("hello.wasm")
    .then(response => response.arrayBuffer())
    .then(buffer => WebAssembly.instantiate(buffer, importObject))
    .then(object => {
        let inputs = ...;
        let result = object.instance.exports.somefun(inputs);
    });
```
WebAssembly API in JavaScript

- WebAssembly global object

- Usual lifecycle:
  - Load .wasm file with fetch()
  - Instantiate the file = compile and provide imports
  - Call exported functions on the instance

- Instantiation methods
  - Blocking: new WebAssembly.Instance(...)
  - Async: await WebAssembly.instantiate(buffer, imports)
  - Streaming: await WebAssembly.instantiateStreaming(fetch(...), imports)
Tooling and Documentation

- **Emscripten: C → WebAssembly compiler**
  - [https://emscripten.org/](https://emscripten.org/)

- **WebAssembly Binary Toolkit**
  - [https://github.com/WebAssembly/wabt](https://github.com/WebAssembly/wabt)
  - wat2wasm
  - wasm2wat
  - wasm-objdump

- **WebAssembly Specification**
  - [https://webassembly.github.io/spec/](https://webassembly.github.io/spec/)
  - E.g., instructions: [https://webassembly.github.io/spec/core/bikeshed/index.html#instructions](https://webassembly.github.io/spec/core/bikeshed/index.html#instructions)
Setup for Hands-On Tasks

- Download VirtualBox [https://www.virtualbox.org/](https://www.virtualbox.org/)
- Copy or download VM image
  - USB stick
- “Import Appliance”
- $ cd ~/tutorial/wasabi/
  
  $ git pull
  
  $ cd ./tutorial/
Task 0.1: A Minimal WebAssembly Program

- **Goals**
  - Understand the text format
  - Convert between text and binary format
  - Embed a WebAssembly binary in a website
  - Run it in the browser

- **Prerequisites**
  - WebAssembly Binary Toolkit: [https://github.com/WebAssembly/wabt](https://github.com/WebAssembly/wabt)

- **Instructions**
  - See README in [https://github.com/danleh/wasabi/tree/master/tutorial/task0/1-hello](https://github.com/danleh/wasabi/tree/master/tutorial/task0/1-hello)
Task 0.2: Addition in WebAssembly

● **Goals**
  ○ Write a more useful function in WebAssembly
  ○ How to call WebAssembly functions from JavaScript
  ○ Bonus: WebAssembly i32 != JavaScript number

● **Prerequisites**
  ○ WebAssembly Binary Toolkit: [https://github.com/WebAssembly/wabt](https://github.com/WebAssembly/wabt)

● **Instructions**
  ○ See README in [https://github.com/danleh/wasabi/tree/master/tutorial/task0/2-add](https://github.com/danleh/wasabi/tree/master/tutorial/task0/2-add)
Task 0.3: Compiling C to WebAssembly

● Goals
  ○ Learn about Emscripten
  ○ Compile C code to WebAssembly + HTML harness with Emscripten
  ○ See that you can “run C code in the browser”

● Prerequisites
  ○ Emscripten: https://emscripten.org/docs/getting_started/downloads.html

● Instructions
  ○ See README in https://github.com/danleh/wasabi/tree/master/tutorial/task0/3-hello-c