WebAssembly

Embedders, Modules, WABT, Primitives, Functions, Linear Memory, Tables, Control Flow, JS API

All the major browser vendors - Google, Mozilla, Microsoft and Apple - have cooperated to define an agreed standard called WebAssembly that specifies how an executable looks like for embedders to execute. WebAssembly allows you to run code written directly in assembly or in a high-level language (e.g. C/C++) compiled into assembly in a browser without plug-ins.

WebAssembly is an intermediate representation (IR) that runs in an embedder. Currently the most popular embedder is the modern standard web browser and on the server the latest Node.js 9 supports it. Other more specialist embedders include the OCaml based spec interpreter and the C++ based WABT interpreter.

WebAssembly modules are binary, low-level (e.g. support 64-bit integers, unlike JavaScript), and are very fast both to load and run. Support for calling WebIDL-defined APIs (e.g. the DOM) will be added in future.

Web developers with requirements for very high performing applications will benefit from exploring WebAssembly as it the basis for considerable industry innovation and has wide cross-browser support.

Target Audience

Developers wishing to get the very best performance out of their web browser and web server code

Prerequisites

Knowledge of C/C++ /JavaScript/TypeScript with some background experience of working with assembly

Contents of One-Day Training Course

WebAssembly Overview

Part of the modern web platform
Designed for the web (security, etc.)
Tour of WebAssembly concepts

WebAssembly Modules

Binary files with .wasm suffix
Can be written by hand (assembly programming) or compiled from C/C++
Overview of assembly syntax
S-Expressions

Embedder

Role of embedders
Embedding in web browsers
Server (Node.js 9)
Custom applications
Spec and WABT

WABT – WebAssembly Binary Toolkit

Low-level CLI tools to work with wasm modules
Assembler, disassembler, interpreter, linker, extractor etc.

Primitive Types

Just four!!
- i32 / i64 / f32 / f64
Working with primitives
Primitive operations

Functions

Defining and calling functions
Function parameters and return value
The call opcode
The start function
Importing/exporting functions

Linear Memory

Handling strings in linear memory
Memory imports and exports
Data section

Structured Control Flow

Hierarchical targets
Don't jump to specific address, rather move up levels in hierarchy
block, loop, if/else, br_table opcodes

Tables

Indirect function calls and security
Tables and the call_indirect opcode
Table imports and exports

JavaScript API

Think of a WebAssembly module as an intermediate representation that is passed to web browser for internal/local compilation
Calling from JS to wasm
Calling from wasm to JS

Module Binary Format

Binary format with well-defined layout
Sections are either named or indexed
Extensible – can add custom sections
LEB128
Opcodes

Future Enhancements

First version called MVP
Future enhancements coming in areas such as SIMD, exceptions, shared buffers, garbage collection and synchronization

Project

Using all the major constructs of WebAssembly in a large project