

# WEBASSEMBLY

## RUNNING C++ AND RUST IN THE BROWSER

Hendrik Niemeyer ([@hniemeye](#))

# LINK TO SLIDES AND CODE

<https://git.io/JvRan>

# MOTIVATION



**Solomon Hykes**

@solomonstre

Folgen



If WASM+WASI existed in 2008, we wouldn't have needed to create Docker. That's how important it is. Webassembly on the server is the future of computing. A standardized system interface was the missing link. Let's hope WASI is up to the task!



**Lin Clark** @linclark

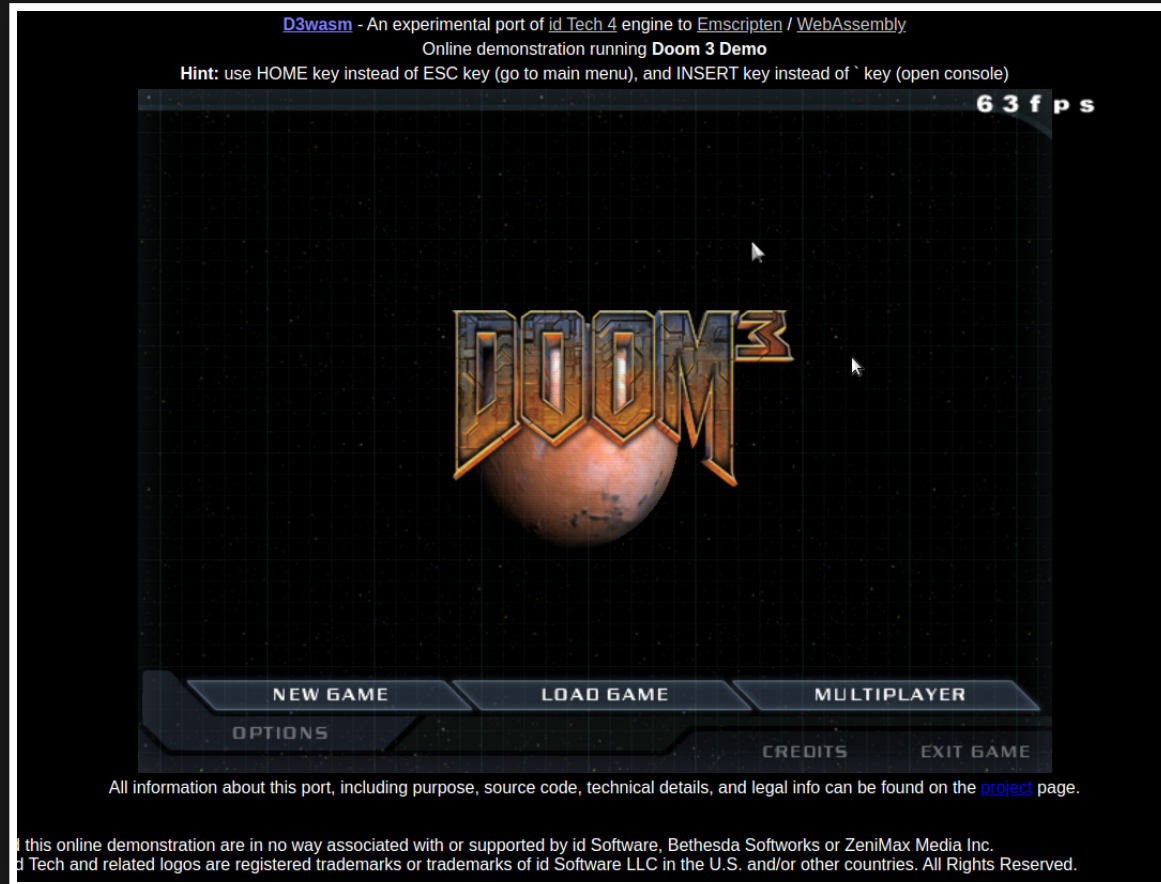
WebAssembly running outside the web has a huge future. And that future gets one giant leap closer today with...

 Announcing WASI: A system interface for running WebAssembly...

[Diesen Thread anzeigen](#)

13:39 - 27. März 2019

# WHAT CAN WEBASSEMBLY DO?



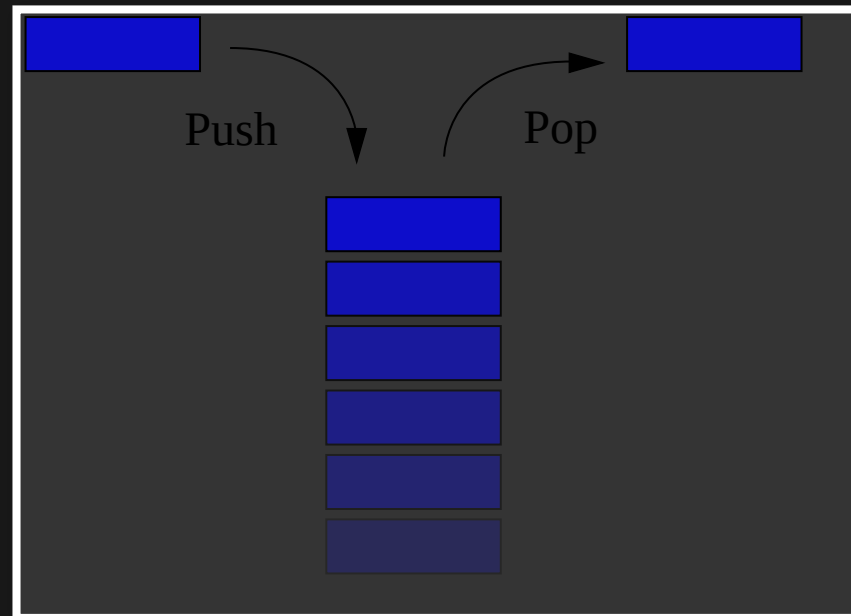
Doom

# WHAT IS WEBASSEMBLY?

*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. (Source: [webassembly.org](https://webassembly.org))*

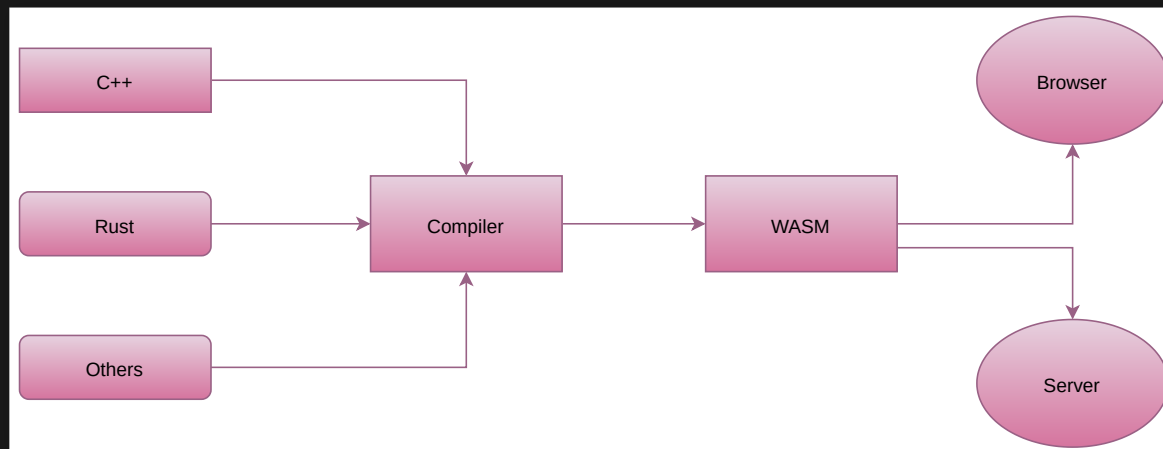
# WHAT IS WEBASSEMBLY?

*WebAssembly (abbreviated Wasm) is a binary instruction format for a **stack-based virtual machine**...*



# WHAT IS WEBASSEMBLY?

*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.*



# OPEN STANDARD

**WebAssembly Core Specification**

W3C Recommendation, 5 December 2019



WebAssembly Core Specification



# USE CASES

- Porting an existing codebase to the browser
- Writing performant code for the browser
- A portable and secure compilation target outside of the browser

# WHAT IS WEBASSEMBLY NOT?




- Not a programming language
- Not intended to replace JavaScript

# WHERE IS IT EXECUTED?













# LANGUAGE SUPPORT

## Legend

-  - Work in progress.
-  - Unstable but usable.
-  - Stable for production usage.

## Contents

-  .Net
-  AssemblyScript
-  Astro
-  Brainfuck
-  C
-  C#
-  C++
-  Clean
-  D
-  Elixir
-  F#
-  Faust
-  Forest
-  Forth
-  Go
-  Grain
-  Haskell
-  Java
-  JavaScript
-  Julia
-  Idris
-  Kotlin/Native
-  Kou
-  Lobster
-  Lua
-  Lys
-  Nim
-  Ocaml
-  Perl
-  PHP
-  Plorth

# LANGUAGE SUPPORT

- Curated list of languages
- No builtin garbage collector
- Languages need to compile their own gc to wasm

# STACK-BASED VIRTUAL MACHINE

```
int add(int a, int b) {  
    return a+b*3;  
}
```

becomes

```
1 (func $func0(param $var0 i32)(param $var1 i32)(result i32)  
2 get_local $var1  
3 i32.const 3  
4 i32.mul  
5 get_local $var0  
6 i32.add  
7 )
```

# STACK-BASED VIRTUAL MACHINE

```
1 (func $func0(param $var0 i32)(param $var1 i32)(result i32)
2 get_local $var1
3 i32.const 3
4 i32.mul
5 get_local $var0
6 i32.add
7 )
```



# STACK-BASED VIRTUAL MACHINE

```
1 (func $func0(param $var0 i32)(param $var1 i32)(result i32)
2  get_local $var1
3  i32.const 3
4  i32.mul
5  get_local $var0
6  i32.add
7  )
```





# STACK-BASED VIRTUAL MACHINE

```
1 (func $func0(param $var0 i32)(param $var1 i32)(result i32)
2 get_local $var1
3 i32.const 3
4 i32.mul
5 get_local $var0
6 i32.add
7 )
```



# STACK-BASED VIRTUAL MACHINE

```
1 (func $func0(param $var0 i32)(param $var1 i32)(result i32)
2 get_local $var1
3 i32.const 3
4 i32.mul
5 get_local $var0
6 i32.add
7 )
```

3\*var1

# STACK-BASED VIRTUAL MACHINE

```
1 (func $func0(param $var0 i32)(param $var1 i32)(result i32)
2 get_local $var1
3 i32.const 3
4 i32.mul
5 get_local $var0
6 i32.add
7 )
```

var0

3\*var1

# STACK-BASED VIRTUAL MACHINE

```
1 (func $func0(param $var0 i32)(param $var1 i32)(result i32)
2 get_local $var1
3 i32.const 3
4 i32.mul
5 get_local $var0
6 i32.add
7 )
```

var0+3\*var1



# TYPES

- i32, i64 are 32/64-bit integer
- f32, f64 are 32/64-bit floating point

# OTHER TYPES?

- What becomes of structs, arrays, vectors, ... ?

# MEMORY

```
struct Point {  
    double x;  
    double y;  
};  
  
double scalar_product(const Point& p1, const Point& p2) {  
    return p1.x*p2.x + p1.y*p2.y;  
}
```

# MEMORY

becomes

```
1 (func $func0(param $var0 i32)(param $var1 i32)(result f64)
2 get_local $var0
3 f64.load
4 get_local $var1
5 f64.load
6 f64.mul
7 get_local $var0
8 f64.load offset=8
9 get_local $var1
10 f64.load offset=8
11 f64.mul
12 f64.add
13 )
```



# MEMORY

becomes

```
1 (func $func0(param $var0 i32)(param $var1 i32)(result f64)
2 get_local $var0
3 f64.load
4 get_local $var1
5 f64.load
6 f64.mul
7 get_local $var0
8 f64.load offset=8
9 get_local $var1
10 f64.load offset=8
11 f64.mul
12 f64.add
13 )
```

# MEMORY

becomes

```
1 (func $func0(param $var0 i32)(param $var1 i32)(result f64)
2  get_local $var0
3  f64.load
4  get_local $var1
5  f64.load
6  f64.mul
7  get_local $var0
8  f64.load offset=8
9  get_local $var1
10 f64.load offset=8
11 f64.mul
12 f64.add
13 )
```

# MEMORY

becomes

```
1 (func $func0(param $var0 i32)(param $var1 i32)(result f64)
2 get_local $var0
3 f64.load
4 get_local $var1
5 f64.load
6 f64.mul
7 get_local $var0
8 f64.load offset=8
9 get_local $var1
10 f64.load offset=8
11 f64.mul
12 f64.add
13 )
```

# MEMORY

becomes

```
1 (func $func0(param $var0 i32)(param $var1 i32)(result f64)
2 get_local $var0
3 f64.load
4 get_local $var1
5 f64.load
6 f64.mul
7 get_local $var0
8 f64.load offset=8
9 get_local $var1
10 f64.load offset=8
11 f64.mul
12 f64.add
13 )
```

# MEMORY

becomes

```
1 (func $func0(param $var0 i32)(param $var1 i32)(result f64)
2 get_local $var0
3 f64.load
4 get_local $var1
5 f64.load
6 f64.mul
7 get_local $var0
8 f64.load offset=8
9 get_local $var1
10 f64.load offset=8
11 f64.mul
12 f64.add
13 )
```

# MEMORY

becomes

```
1 (func $func0(param $var0 i32)(param $var1 i32)(result f64)
2 get_local $var0
3 f64.load
4 get_local $var1
5 f64.load
6 f64.mul
7 get_local $var0
8 f64.load offset=8
9 get_local $var1
10 f64.load offset=8
11 f64.mul
12 f64.add
13 )
```

# LINEAR MEMORY

- contiguous, byte-addressable range of memory
- untyped array of bytes
- load and store instructions for floats and integers

# ARRAYS

```
void change_fifth(std::array<int,10>& arr, int n)
{
    arr[5] = n - 5;
}
```

becomes

```
1 (func $func0 (param $var0 i32) (param $var1 i32)
2 get_local $var0
3 get_local $var1
4 i32.const -5
5 i32.add
6 i32.store offset=20
7 )
```



# ARRAYS

```
void change_fifth(std::array<int,10>& arr, int n)
{
    arr[5] = n - 5;
}
```

becomes

```
1 (func $func0 (param $var0 i32) (param $var1 i32)
2 get_local $var0
3 get_local $var1
4 i32.const -5
5 i32.add
6 i32.store offset=20
7 )
```

# ARRAYS

```
void change_fifth(std::array<int,10>& arr, int n)
{
    arr[5] = n - 5;
}
```

becomes

```
1 (func $func0 (param $var0 i32) (param $var1 i32)
2  get_local $var0
3  get_local $var1
4  i32.const -5
5  i32.add
6  i32.store offset=20
7  )
```

# ARRAYS

```
void change_fifth(std::array<int,10>& arr, int n)
{
    arr[5] = n - 5;
}
```

becomes

```
1 (func $func0 (param $var0 i32) (param $var1 i32)
2  get_local $var0
3  get_local $var1
4  i32.const -5
5  i32.add
6  i32.store offset=20
7  )
```

# ARRAYS

```
void change_fifth(std::array<int,10>& arr, int n)
{
    arr[5] = n - 5;
}
```

becomes

```
1 (func $func0 (param $var0 i32) (param $var1 i32)
2 get_local $var0
3 get_local $var1
4 i32.const -5
5 i32.add
6 i32.store offset=20
7 )
```

# CONTROL FLOW

- `block ... end`: Sequence of instructions with label at the end
- `loop ... end`: Sequence of instructions with label at the beginning
- Comparison instructions like `f32.lt`
- `br, br_if`: jump (conditionally) to a given label in an enclosing construct

# WEBASSEMBLY EXPLORER

# COMPILING C++ TO WASM DIRECTLY

```
clang++ \  
  --target=wasm32 \  
  -nostdlib \ # Don't link against a standard library  
  -Wl,--no-entry \ # We don't have an entry function  
  -Wl,--export-all \ # Export everything  
  -o add.wasm \  
  add.cpp
```

# LOADING WASM INTO THE BROWSER

```
<script type="module">
async function init() {
const { instance } = await WebAssembly.instantiateStreaming(
  fetch("./add.wasm")
);
console.log(instance.exports._Z3addii(4, 1));
const pre = document.getElementById("my-canvas");
pre.textContent = instance.exports._Z3addii(4, 1)

}
init();
</script>
```



# COMPILING C++ TO WASM WITH COMPILER EXPLORER

<https://godbolt.org/z/6pF8pm>

# C++ TO WEBASSEMBLY



***emscripten***

# EMSCRIPTEN

```
em++ -o hello.html hello.cpp
```

# RUST TO WEBASSEMBLY



# WASMPACK

```
wasm-pack build  
cd www/ || exit  
npm install  
npm run start
```

# RUN DIRECTLY ON THE OS



# QUESTIONS AND FEEDBACK

- Twitter: [@hniemeye](#)
- LinkedIn: [www.linkedin.com/in/hniemeyer87](http://www.linkedin.com/in/hniemeyer87)