Clang! Clang! Who’s there? WebAssembly!

Paulo Matos (pmatos@igalia.com)
Alex Bradbury (asb@igalia.com)
Andy Wingo (wingo@igalia.com)

November 9, 2022

Thanks to

igalia My teammates Alex Bradbury and Andy Wingo!

The Wasm Tools Team for suggestions, discussions and reviews! Google

Bloomberg For Sponsoring this work!

WebAssembly for those in a Rush

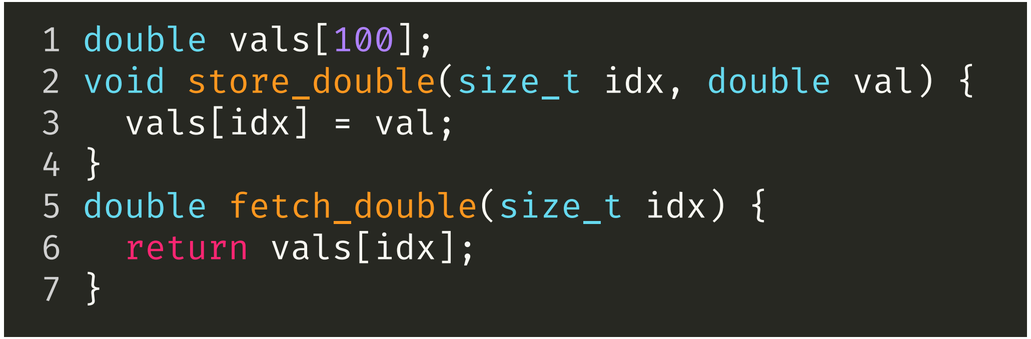
The WebAssembly VM (spec: https://webassembly.github.io/spec/core/)

- Harvard Architecture (Linear memory separate from code)
- Well-typed functions and globals organized into modules
- Module-level globals and Function-level locals not addressable

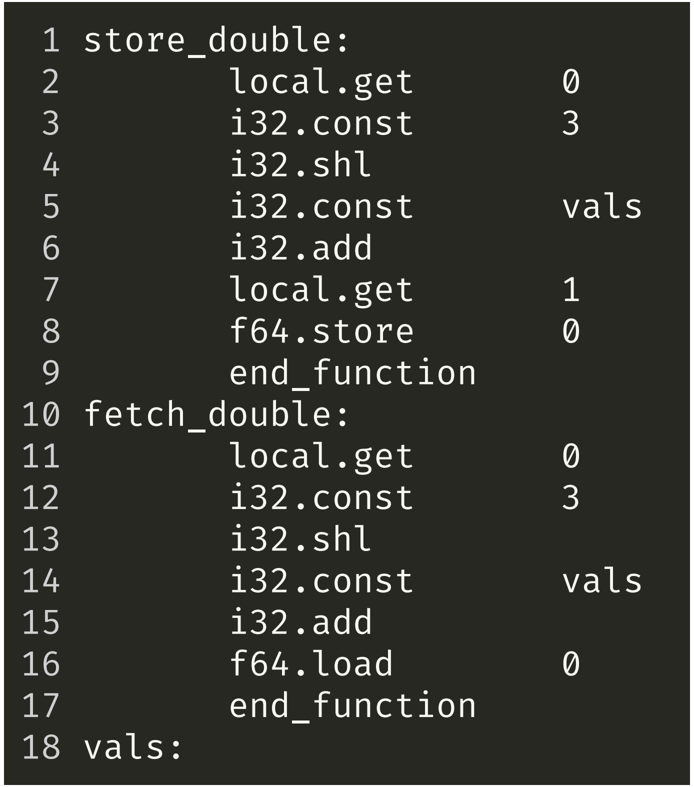
Initially:

- 4 numeric types { i32, i64, f32, f64 }
- Spec provides instructions to manipulate values of these types
- Load-store instructions to manipulate memory

Compile to Wasm



```c
double vals[100];
void store_double(size_t idx, double val) {
  vals[idx] = val;
}
double fetch_double(size_t idx) {
  return vals[idx];
}
```

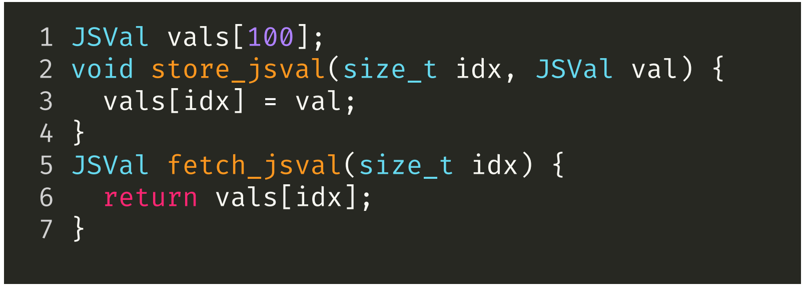


```
store_double:
        local.get       0
        i32.const       3
        i32.shl
        i32.const       vals
        i32.add
        local.get       1
        f64.store       0
        end_function
fetch_double:
        local.get       0
        i32.const       3
        i32.shl
        i32.const       vals
        i32.add
        f64.load        0
        end_function
vals:
```

https://godbolt.org/z/r4WMncWKb

Evolving WebAssembly

What if instead of doubles we have JS values?



```c
JSVal vals[100];
void store_jsval(size_t idx, JSVal val) {
  vals[idx] = val;
}
JSVal fetch_jsval(size_t idx) {
  return vals[idx];
}
```

Doesn’t work for reftypes!

Enter Reference Types:

- These types are **opaque and host-managed**
- Can’t be stored to linear memory
  - but they can be arguments and return values from functions
  - can be stored to globals
- Currently *two different types*: funcref and externref.
  - funcref is a callable reference type

The main challenge we are trying to solve is ***how to represent these values internally in Clang and LLVM but also how to expose them to the user as C/C++ extensions.***

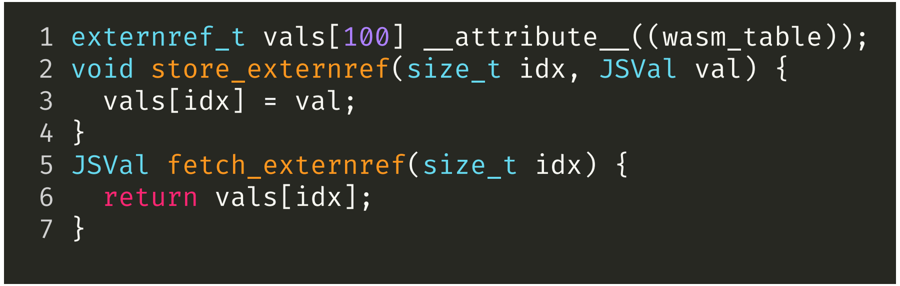
WebAssembly Tables

Tables are used to store reference types

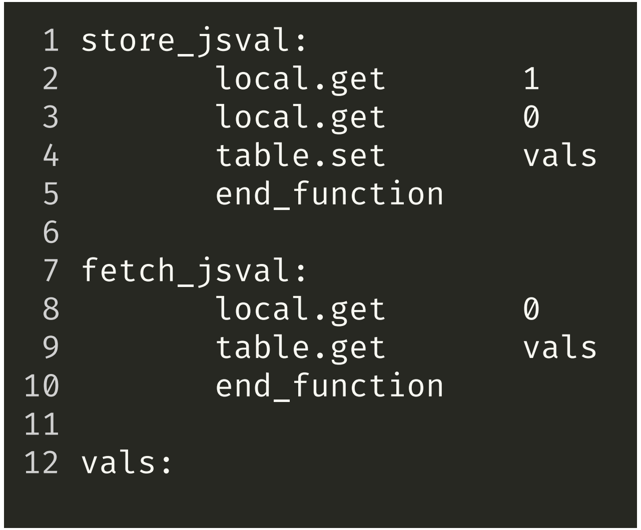
- Even weirder than the reftypes themselves!

They have a bunch of constraints:

- Can’t be stored to linear memory, or stack!
- Can’t be arguments or return values of functions!
- They are global static values in a module.



```
externref_t vals[100] __attribute__((wasm_table));
void store_externref(size_t idx, JSVal val) {
  vals[idx] = val;
}
JSVal fetch_externref(size_t idx) {
  return vals[idx];
}
```



```
store_jsval:
        local.get       1
        local.get       0
        table.set       vals
        end_function

fetch_jsval:
        local.get       0
        table.get       vals
        end_function

vals:
```

Reference Types in LLVM IR

🥳 Support for Reference Types (including tables has landed).

```
%externref = type ptr addrspace(10)
%funcref = type ptr addrspace(20)
@table = local_unnamed_addr addrspace(1) global [0 x %funcref] undef
```

- Addrspace 1, 10, 20 are non-integral
- Tables are represented as global arrays and accessed via intrinsics.

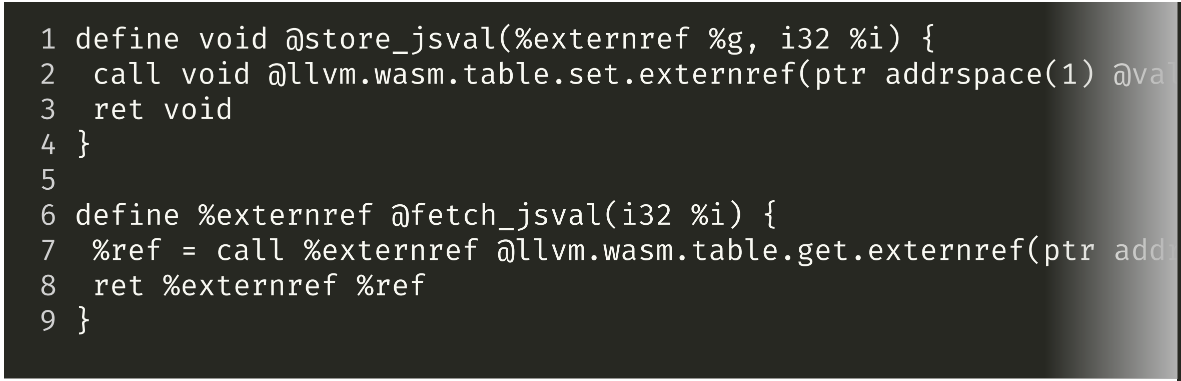
⇩

`MVT::externref`

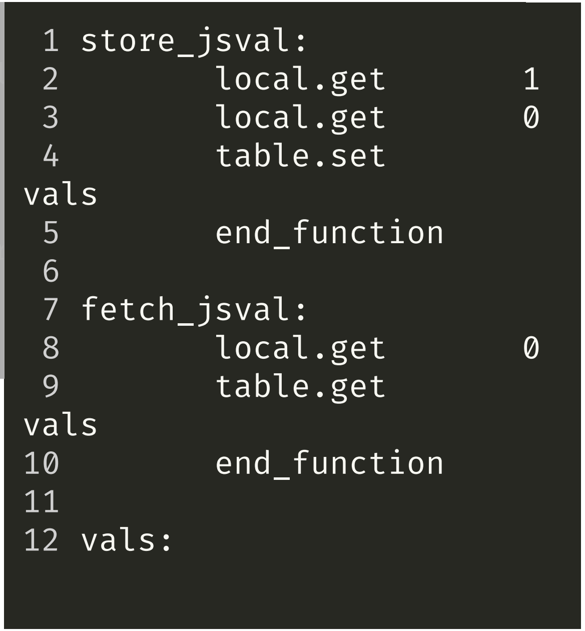
addrspace(1) for object that don't have an in-memory representation

`MVT::funcref`

Reference Types in LLVM IR



```
define void @store_jsval(%externref %g, i32 %i) {
 call void @llvm.wasm.table.set.externref(ptr addrspace(1) @va
 ret void
}

define %externref @fetch_jsval(i32 %i) {
 %ref = call %externref @llvm.wasm.table.get.externref(ptr add
 ret %externref %ref
}
```



```
store_jsval:
        local.get       1
        local.get       0
        table.set
vals
        end_function

fetch_jsval:
        local.get       0
        table.get
vals
        end_function

vals:
```

https://godbolt.org/z/jnb8x93a4

What's the story in Clang?

Reference Types in Clang

In Clang we need:

- a syntax to represent reference types and
- lowering to LLVM IR

Currently being worked out in a downstream public branch (initial prototypes as D122215, D128440, D123510, D124162)

`__externref_t` is a new type

- with the expected reftype constraints
- lowered to LLVM IR as a ptr to addrspace(10)

```
__externref_t JSVAL;
```

funcref is dealt with differently!

- attribute is attached to function pointers: `__funcref`
- lowered to LLVM IR as a ptr to addrspace(20)

```
typedef void (*__funcref fn_vv_t)();
typedef int (*__funcref fn_ii_t)(int);
```

Reference Types in Clang

Tables store reference types!

- Given it’s indexed by an integer, sounds like an array representation is best.
    - However, internally representing a table indexing as `ArraySubscript` causes issues with over-optimization!
      `foo[i] ~> ArraySubscript`
- Alternatively, implement a new AST node `TableSubscript`
    - own set of problems as it requires new debug information impl, ABI info, etc.
      `foo[i] ~> TableSubscript (new)`
- Simpler alternative is to model tables and operations as Intrinsics.
    - Syntax not as ergonomic but possibly quicker path to goal and easier to upstream (?).
      ~~`foo[i]`~~ `instead __wasm_table_get(foo, i)`

Have patch for first approach - wip patch for second - trying out third approach! 😅

WebAssembly GC

Reference types were just a taster for what's to come!

- GC managed objects: arrays and structs
  - (this is unrelated to GC support in LLVM)
- New instructions introduced to manipulate this type
  - i.e access struct fields, array access, etc
- A lot of the work we are doing at the moment is understanding how to represent these in LLVM!
  - but we want to take this all the way to Clang!

Problem: *Current AS approach won't scale for all GC types!*

Need to produce correctly typed locals, globals, function argos and returns

- WebAssembly GC types need to be maintained from LLVM IR through to the backend.

This includes parameterised types, typed function references, etc.

- Therefore defining a new MVT for each won't work.

WebAssembly GC

Our current approach:

- it’s key that type identity is maintained
- Approach is to have one AS ID (currently > 255) as an index into a metadata
- Module-level metadata table index by AS ID tracks value types

The goal is to pass these types through LLVM IR all the way to WebAssembly emission as we cannot translate these to LLVM’s limited type system.

This approach is meant as a prototype - not the one we intend to use going forwards. Hopefully we can work together upstream to find a better solution for IR-level opaque type support!

WebAssembly GC Example 1

```
!0 = !{!"externref"}
!wasm.type_info = !{!0}

%wasmref = type ptr addrspace(256)
%externref = type ptr addrspace(257)

@externref_table = local_unnamed_addr addrspace(1) global [0 x %externref] undef

declare %wasmref @llvm.wasm.table.get.wasmref(ptr addrspace(1), i32) nounwind

define %externref @get_externref_from_table(i32 %i) {
  %ref_u = call %wasmref @llvm.wasm.table.get.wasmref(ptr addrspace(1) @externref_table, i32 %i)
  %ref = addrspacecast %wasmref %ref_u to %externref
  ret %externref %ref
}
```

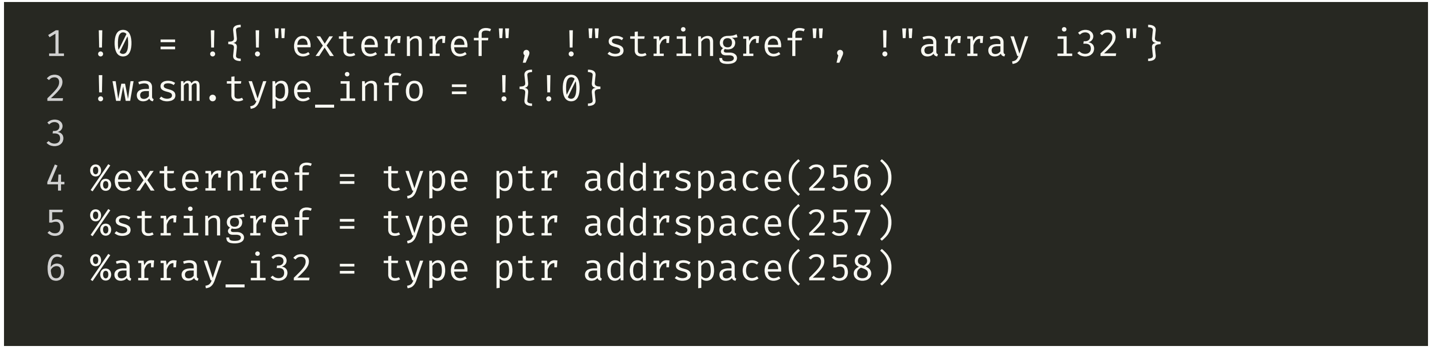
WebAssembly GC Example 2

```
!0 = !{!"array i32"}
!wasm.type_info = !{!0}
%array_i32 = type ptr addrspace(257)

%alloca_cell = type ptr addrspace(1)

declare void @inhibit_store_to_load_forwarding()

define %array_i32 @ir_local_array_i32(%array_i32 %arg) {
 %retval = alloca %array_i32, addrspace(1)
 store %array_i32 %arg, %alloca_cell %retval
 call void @inhibit_store_to_load_forwarding()
 %reloaded = load %array_i32, %alloca_cell %retval
 ret %array_i32 %reloaded
}
```

Lets Talk Strings - stringref

Strings as GC reference types!

- Need to support GC strings across the toolchain
  - currently focusing on LLVM atm
- Using the same mechanism as other Wasm GC types, where stringref would inherit an AS ID.



```
!0 = !{!"externref", !"stringref", !"array i32"}
!wasm.type_info = !{!0}

%externref = type ptr addrspace(256)
%stringref = type ptr addrspace(257)
%array_i32 = type ptr addrspace(258)
```

Summary

- ✅ Reference types in LLVM IR
- 👷🏽 Reference types in Clang (D122215, D128440, D123510, D124162)
- 👷🏽 GC types in LLVM IR (public branch downstream)
- 👷🏽 Stringref in LLVM IR (public branch downstream)
- 🤔 GC types in Clang
- 🤔 Stringref in Clang

We are hiring!
https://www.igalia.com/jobs/