A walk on the weird side

Opaque WebAssembly values and LLVM

LLVM Developers' Meeting 2021

Andy Wingo | *wingo@igalia.com*

Paulo Matos | *pmatos@igalia.com*

Agenda

A quick recap on how LLVM targets WebAssembly

A spanner in the works: opaque host-managed values

LLVM on the weird side

The WebAssembly machine

Harvard architecture: linear memory for data, separated from code

Code structured as well-typed functions

Host manages call stack; return addrs inaccessible

Local variables not addressable

`https://webassembly.github.io/spec/core/syntax/types.html`

numtype ::= `i32` | `i64` | `f32` | `f64`

valtype ::= *numtype*

functype ::= *valtype** ⇒ *valtype**

func := {type: *functype*, locals: *valtype**, body: *expr*}

module := {funcs: *func**, memories: *mem**, imports: *functype**, exports: *func-index**...}

The LLVM WebAssembly target

```
void f() { int i = g(); ... }
```

```
%iloc = alloca i32
%t = call i32 @g()
store i32 %t, i32* %iloc
```

(SROA may eliminate `%iloc`)

SSA variables lower as “managed” WebAssembly function locals and temporaries

```
.functype g () -> (i32)

f:
   .functype f () -> ()
   .local i32
   call g
   local.set 0
   ...
```

Otherwise `alloca` lowers as stack allocation - linear memory partitioned into stack, data, heap

New developments in WebAssembly

`https://webassembly.github.io/spec/core/syntax/types.html`

numtype ::= `i32` | `i64` | `f32` | `f64`

reftype **::= `externref` | `funcref`**

valtype ::= *numtype* | ***reftype***

functype ::= *valtype** ⇒ *valtype**

func ::= {type: *functype*, locals: *valtype**, body: *expr*}

`externref` (& `funcref`) opaque, host-managed values

Only *numtype* can be stored to linear memory, not *reftype*

Use: GC-managed data, host objects (`FILE*`, JavaScript, …)

New problems for LLVM

- IR: Types, SSA values, intrinsics, lowering
- Clang: Alloca, the user experience

Compiler writers' full employment act still in effect

IR & externref

The hack at the heart of things:

```
%extern = type opaque
%externref = type %extern addrspace(10)*

MVT getPointerTy(const DataLayout &DL,
                 uint32_t AS = 0) const override {
  if (AS == WasmAddressSpace::EXTERNREF)
    return MVT::externref;
  if (AS == WasmAddressSpace::FUNCREF)
    return MVT::funcref;
  return TargetLowering::getPointerTy(DL, AS);
}
```

Horribly effective

How to program externref?

```
void f() { externref_t v = g(); }
```

```
%vloc = alloca ? ; ??????
```

```
struct { int a; externref_t b; } v; // ?
```

Not obvious!

Smells like SVE spirit...

Scalable vectors (e.g. SVE) introduce notion of sizeless types: same restrictions on use as incomplete types, plus a couple others

Semantics defined in ARM C Language Extensions (ACLE)

Sema applies ACLE restrictions for SVE values

`externref` piggy-backs on these restrictions

...with subtle notes of weird

SVE values do have byte representation, `externref` does not

Not “unknown size in memory” but rather “can’t be put in linear memory”

No `inttoptr` will ever address an `externref`; `ptrtoint` meaningless

New restriction for frontend: no address-of on reference types

```
%vloc = alloca ? ; ??????
```

Our solution

Another friggin address space

```
%iloc = alloca i32, addrspace(0)

%extern = type opaque
%externref = type %extern addrspace(10)*
%vloc = alloca %externref, addrspace(1)
```

SROA can still lift to SSA

If static `alloca` in addrspace 1 reaches backend, it lowers to (mutable) function local, not linear memory

Clang never issues non-static `alloca` in addrspace 1

Initial idea of exposing “wasm function local address space” to user inspired by e.g. `__global` from OpenCL – but unnecessarily broad

- Current approach is typed-based
- Some points remaining to iron out

Status

IR: Done, essentially

Clang: In progress. Delicate.

Goal: Allow host gc-managed aggregate types; leak-free JS/C++ systems in web runtimes

Thanks for listening, and for reviewer patience for a very weird machine!

`https://github.com/Igalia/ref-cpp`