Neither Web nor Assembly

Andreas Rossberg
Dfinity

A portable code format

Bringing native code performance

To browsers near you

A virtual instruction set architecture

That is fully sandboxed

And can be embedded everywhere

open standard (W3C, github)

not proprietary, copyrighted, or

Why?

High performance (within 10% of native code)

Predictable performance

Empower other languages than JavaScript

Enable features that JavaScript can't

Supersede asm.js and (P)NaCl

Goals & Constraints

Semantics

- Language-independent
- Platform-independent
- Hardware-independent
- Fast to execute
- Safe to execute
- Deterministic
- Easy to reason about

Representation

- Compact
- Easy to generate
- Fast to decode
- Fast to validate
- Fast to compile
- Streamable
- Parallelisable

byte code

hardware-independent

designed for jitting

stack machine

most compact

stack layout is entirely static

hardware types & operators

int32, int64, float32, float64

operators common to modern CPUs

linear memory

just a byte array, integers as pointers

bounds-checked, growable

no built-in objects!

structured control flow

blocks and breaks, no arbitrary jumps

fast to validate & compile

producers use *relooper* algorithm if necessary
[Zakai, OOPSLA 2011]

blocks can have results, branches can take arguments

type checking

type-safe and memory-safe

trusted stack

efficient execution

modular & sandboxed

binaries are modules

sandboxed, no ambient capabilities

imports can be host functions
(on the web, JavaScript as an FFI)

```
(module
  (func $fac (param $x i64) (result i64)      00
    (get_local $x)                            23 00
    (i64.eqz)                                 50
    (if                                       04
      (then (i64.const 1))                    42 01
      (else                                   05
        (get_local $x)                        23 00
        (i64.const 1)                         42 01
        (i64.sub)                             7D
        (call $fac)                           10 00
        (get_local $x)                        23 00
        (i64.mul)                             7E
      )                                       0B
    )                                         0B
  )
  (export “fac” (func $fac))
)
```

```
fac(x : int64) : int64 =
  if not x
  then 1
  else x * fac(x - 1)
```

wasm meets formal methods

completely formal semantics

integral part of the design process

catch up with the last 50 years of PL research

meta-goal: raise the bar for industrial language design

(value types)	t	$::=$	i32 | i64 | f32 | f64
(packed types)	pt	$::=$	i8 | i16 | i32
(function types)	ft	$::=$	$t^* \rightarrow t^*$
	$unop$	$::=$	**neg** | **abs** | ...
	$binop$	$::=$	**add** | **sub** | **mul** | **div_s** | **div_u** | ...
	$relop$	$::=$	**eq** | **ne** | **lt** | **gt** | ...
	$cvtop$	$::=$	**convert/**t | **reinterpret/**t
(instructions)	e	$::=$	t**.const** c | $t.unop$ | $t.binop$ | $t.relop$ | $t.cvtop$ |
			unreachable | **nop** | **drop** | **select** |
			block $ft\ e^*$ **end** | **loop** $ft\ e^*$ **end** | **if** $ft\ e^*$ **else** e^* **end** |
			br i | **br_if** i | **br_table** $i^*\ i$ |
			call i | **call_indirect** ft | **return** |
			get_local i | **set_local** i | **tee_local** i |
			get_global i | **set_global** i |
			t**.load** $pt^?\ n$ | t**.store** $pt^?\ n$ | **current_mem** | **grow_mem**
(functions)	$func$	$::=$	**func** ft (**local** $t)^*\ e^*$
(globals)	$glob$	$::=$	**global mut**$^?\ t\ e^*$
(tables)	tab	$::=$	**table** $n\ i^*$
(memories)	mem	$::=$	**memory** n
(modules)	m	$::=$	**module** $import^*\ func^*\ glob^*\ tab^?\ mem^?\ export^*$

operational semantics

standard small-step reduction rules

deterministic (up to NaN bits)

no undefined behaviour

(store)	s	$::=$	$\{\textsf{inst}\ \mathit{inst}^*,\ \textsf{tab}\ \mathit{tabinst}^*,\ \textsf{mem}\ \mathit{meminst}^*\}$
(instances)	inst	$::=$	$\{\textsf{func}\ cl^*,\ \textsf{glob}\ v^*,\ \textsf{tab}\ i^?,\ \textsf{mem}\ i^?\}$
	$\mathit{tabinst}$	$::=$	cl^*
	$\mathit{meminst}$	$::=$	b^*
(closures)	cl	$::=$	$\{\textsf{inst}\ i,\ \textsf{code}\ f\}$ (where f is not an import and has all exports ex^* erased)
(values)	v	$::=$	$t.\textbf{const}\ c$
(administrative operators)	e	$::=$	$\dots \mid \textbf{trap} \mid \textbf{call}\ cl \mid \textbf{label}\{t^*; e^*\}\ e^*\ \textbf{end} \mid \textbf{local}\{i; v^*\}\ e^*\ \textbf{end}$
(local contexts)	L^0	$::=$	$v^*\ [\_]\ e^*$
	L^{k+1}	$::=$	$v^*\ \textbf{label}\{t^*; e^*\}\ L^k\ \textbf{end}\ e^*$

Reduction $\dfrac{s; v^*; e^* \hookrightarrow_i s'; v'^*; e'^*}{s; v^*; L^k[e^*] \hookrightarrow_i s'; v'^*; L^k[e'^*]}$ $\qquad \dfrac{s; v^*; e^* \hookrightarrow_i s'; v'^*; e'^*}{s; v_0^*; \textbf{local}\{i; v^*\}\ e^*\ \textbf{end} \hookrightarrow_j s'; v_0^*; \textbf{local}\{i; v'^*\}\ e'^*\ \textbf{end}}$ $\qquad \boxed{s; v^*; e^* \hookrightarrow_i s; v^*; e^*}$

$(t.\textbf{const}\ c)\ t.\mathit{unop}$	$\hookrightarrow$	$t.\textbf{const}\ \mathit{unop}_t(c)$	
$(t.\textbf{const}\ c_1)\ (t.\textbf{const}\ c_2)\ t.\mathit{binop}$	$\hookrightarrow$	$t.\textbf{const}\ c$	if $c = \mathit{binop}_t(c_1, c_2)$
$(t.\textbf{const}\ c_1)\ (t.\textbf{const}\ c_2)\ t.\mathit{binop}$	$\hookrightarrow$	$\textbf{trap}$	otherwise
$(t.\textbf{const}\ c)\ t.\mathit{testop}$	$\hookrightarrow$	$\textbf{i32.const}\ \mathit{testop}_t(c)$	
$(t.\textbf{const}\ c_1)\ (t.\textbf{const}\ c_2)\ t.\mathit{relop}$	$\hookrightarrow$	$\textbf{i32.const}\ \mathit{relop}_t(c_1, c_2)$	
$(t_1.\textbf{const}\ c)\ t_2.\textbf{convert}\ t_1\_sx^?$	$\hookrightarrow$	$t_2.\textbf{const}\ c'$	if $c' = \mathrm{cvt}^{sx^?}_{t_1,t_2}(c)$
$(t_1.\textbf{const}\ c)\ t_2.\textbf{convert}\ t_1\_sx^?$	$\hookrightarrow$	$\textbf{trap}$	otherwise
$(t_1.\textbf{const}\ c)\ t_2.\textbf{reinterpret}\ t_1$	$\hookrightarrow$	$t_2.\textbf{const}\ \mathrm{const}_{t_2}(\mathrm{bits}_{t_1}(c))$	
$\textbf{unreachable}$	$\hookrightarrow$	$\textbf{trap}$	
$\textbf{nop}$	$\hookrightarrow$	ϵ	
$v\ \textbf{drop}$	$\hookrightarrow$	ϵ	
$v_1\ v_2\ (\textbf{i32.const}\ 0)\ \textbf{select}$	$\hookrightarrow$	v_2	
$v_1\ v_2\ (\textbf{i32.const}\ k+1)\ \textbf{select}$	$\hookrightarrow$	v_1	
$v^n\ \textbf{block}\ (t_1^n \to t_2^m)\ e^*\ \textbf{end}$	$\hookrightarrow$	$\textbf{label}\{t_2^m; \epsilon\}\ v^n\ e^*\ \textbf{end}$	
$v^n\ \textbf{loop}\ (t_1^n \to t_2^m)\ e^*\ \textbf{end}$	$\hookrightarrow$	$\textbf{label}\{t_1^n; \textbf{loop}\ (t_1^n \to t_2^m)\ e^*\ \textbf{end}\}\ v^n\ e^*\ \textbf{end}$	
$(\textbf{i32.const}\ 0)\ \textbf{if}\ \mathit{tf}\ e_1^*\ \textbf{else}\ e_2^*\ \textbf{end}$	$\hookrightarrow$	$\textbf{block}\ \mathit{tf}\ e_2^*\ \textbf{end}$	
$(\textbf{i32.const}\ k+1)\ \textbf{if}\ \mathit{tf}\ e_1^*\ \textbf{else}\ e_2^*\ \textbf{end}$	$\hookrightarrow$	$\textbf{block}\ \mathit{tf}\ e_1^*\ \textbf{end}$	
$\textbf{label}\{t^*; e^*\}\ v^*\ \textbf{end}$	$\hookrightarrow$	v^*	
$\textbf{label}\{t^*; e^*\}\ \textbf{trap}\ \textbf{end}$	$\hookrightarrow$	$\textbf{trap}$	
$\textbf{label}\{t^n; e^*\}\ L^j[v^n\ (\textbf{br}\ j)]\ \textbf{end}$	$\hookrightarrow$	$v^n\ e^*$	
$(\textbf{i32.const}\ 0)\ (\textbf{br\_if}\ j)$	$\hookrightarrow$	ϵ	
$(\textbf{i32.const}\ k+1)\ (\textbf{br\_if}\ j)$	$\hookrightarrow$	$\textbf{br}\ j$	
$(\textbf{i32.const}\ k)\ (\textbf{br\_table}\ j_1^k\ j\ j_2^*)$	$\hookrightarrow$	$\textbf{br}\ j$	
$(\textbf{i32.const}\ k+n)\ (\textbf{br\_table}\ j_1^k\ j)$	$\hookrightarrow$	$\textbf{br}\ j$	
$s; \textbf{call}\ j$	$\hookrightarrow_i$	$\textbf{call}\ s_{\mathsf{func}}(i, j)$	
$s; (\textbf{i32.const}\ j)\ \textbf{call\_indirect}\ \mathit{tf}$	$\hookrightarrow_i$	$\textbf{call}\ s_{\mathsf{tab}}(i, j)$	if $s_{\mathsf{tab}}(i,j)_{\mathsf{code}} = (\textbf{func}\ \mathit{tf}\ \textbf{local}\ t^*\ e^*)$
$s; (\textbf{i32.const}\ j)\ \textbf{call\_indirect}\ \mathit{tf}$	$\hookrightarrow_i$	$\textbf{trap}$	otherwise
$v^n\ (\textbf{call}\ cl)$	$\hookrightarrow$	$\textbf{local}\{cl_{\mathsf{inst}}; v^n\ (t.\textbf{const}\ 0)^k\}\ \textbf{block}\ (\epsilon \to t_2^m)\ e^*\ \textbf{end}\ \textbf{end}$ ...	
$\textbf{local}\{i; v_l^*\}\ v^*\ \textbf{end}$	$\hookrightarrow$	v^*	| ... if $cl_{\mathsf{code}} = (\textbf{func}\ (t_1^n \to t_2^m)\ \textbf{local}\ t^k\ e^*)$
$\textbf{local}\{i; v_l^*\}\ \textbf{trap}\ \textbf{end}$	$\hookrightarrow$	$\textbf{trap}$	
$\textbf{local}\{i; v_l^*\}\ L^{k+1}[\textbf{return}]\ \textbf{end}$	$\hookrightarrow$	$\textbf{local}\{i; v_l^*\}\ L^{k+1}[\textbf{br}\ k]\ \textbf{end}$	
$v_1^j\ v\ v_2^k; \textbf{get\_local}\ j$	$\hookrightarrow$	v	
$v_1^j\ v\ v_2^k; v'\ (\textbf{set\_local}\ j)$	$\hookrightarrow$	$v_1^j\ v'\ v_2^k; \epsilon$	
$v\ (\textbf{tee\_local}\ j)$	$\hookrightarrow$	$v\ v\ (\textbf{set\_local}\ j)$	
$s; \textbf{get\_global}\ j$	$\hookrightarrow_i$	$s_{\mathsf{glob}}(i, j)$	
$s; v\ (\textbf{set\_global}\ j)$	$\hookrightarrow_i$	$s'; \epsilon$	if $s' = s$ with $\mathsf{glob}(i,j) = v$
$s; (\textbf{i32.const}\ k)\ (t.\textbf{load}\ a\ o)$	$\hookrightarrow_i$	$t.\textbf{const}\ \mathrm{const}_t(b^*)$	if $s_{\mathsf{mem}}(i, k+o, \lvert t\rvert) = b^*$
$s; (\textbf{i32.const}\ k)\ (t.\textbf{load}\ tp\_sx\ a\ o)$	$\hookrightarrow_i$	$t.\textbf{const}\ \mathrm{const}_t^{sx}(b^*)$	if $s_{\mathsf{mem}}(i, k+o, \lvert tp\rvert) = b^*$
$s; (\textbf{i32.const}\ k)\ (t.\textbf{load}\ tp\_sx^?\ a\ o)$	$\hookrightarrow_i$	$\textbf{trap}$	otherwise
$s; (\textbf{i32.const}\ k)\ (t.\textbf{const}\ c)\ (t.\textbf{store}\ a\ o)$	$\hookrightarrow_i$	$s'; \epsilon$	if $s' = s$ with $\mathsf{mem}(i, k+o, \lvert t\rvert) = \mathrm{bits}_t^{\lvert t\rvert}(c)$
$s; (\textbf{i32.const}\ k)\ (t.\textbf{const}\ c)\ (t.\textbf{store}\ tp\ a\ o)$	$\hookrightarrow_i$	$s'; \epsilon$	if $s' = s$ with $\mathsf{mem}(i, k+o, \lvert tp\rvert) = \mathrm{bits}_t^{\lvert tp\rvert}(c)$
$s; (\textbf{i32.const}\ k)\ (t.\textbf{const}\ c)\ (t.\textbf{store}\ tp^?\ a\ o)$	$\hookrightarrow_i$	$\textbf{trap}$	otherwise
$s; \textbf{current\_memory}]$	$\hookrightarrow_i$	$\textbf{i32.const}\ \lvert s_{\mathsf{mem}}(i,*)\rvert/64\,\mathrm{Ki}$	
$s; (\textbf{i32.const}\ k)\ \textbf{grow\_memory}$	$\hookrightarrow_i$	$s'; \textbf{i32.const}\ \lvert s_{\mathsf{mem}}(i,*)\rvert/64\,\mathrm{Ki}$	if $s' = s$ with $\mathsf{mem}(i,*) = s_{\mathsf{mem}}(i,*)\ (0)^{k\cdot 64\,\mathrm{Ki}}$
$s; (\textbf{i32.const}\ k)\ \textbf{grow\_memory}$	$\hookrightarrow_i$	$\textbf{i32.const}\ (-1)$	

Figure 1. Small-step reduction rules

[PLDI 2017]

type system

standard deduction rules

almost embarrassingly simple!

encapsulation, compositional

proof of soundness

$$\text{(contexts)}\quad C \quad ::= \quad \{\mathsf{func}\ tf^*,\ \mathsf{global}\ tg^*,\ \mathsf{table}\ n^?,\ \mathsf{memory}\ n^?,\ \mathsf{local}\ t^*,\ \mathsf{label}\ (t^*)^*\}$$

Typing Instructions $\boxed{C \vdash e^* : tf}$

$$\frac{}{C \vdash t.\mathbf{const}\ c : \epsilon \to t} \quad \frac{}{C \vdash t.\mathit{unop} : t \to t} \quad \frac{}{C \vdash t.\mathit{binop} : t\ t \to t} \quad \frac{}{C \vdash t.\mathit{testop} : t \to \mathsf{i32}} \quad \frac{}{C \vdash t.\mathit{relop} : t\ t \to \mathsf{i32}}$$

$$\frac{t_1 \neq t_2 \qquad sx^? = \epsilon \Leftrightarrow (t_1 = \mathsf{i}n \wedge t_2 = \mathsf{i}n' \wedge |t_1| < |t_2|) \vee (t_1 = \mathsf{f}n \wedge t_2 = \mathsf{f}n')}{C \vdash t_1.\mathbf{convert}\ t_2\_sx^? : t_2 \to t_1} \quad \frac{t_1 \neq t_2 \qquad |t_1| = |t_2|}{C \vdash t_1.\mathbf{reinterpret}\ t_2 : t_2 \to t_1}$$

$$\frac{}{C \vdash \mathbf{unreachable} : t_1^* \to t_2^*} \quad \frac{}{C \vdash \mathbf{nop} : \epsilon \to \epsilon} \quad \frac{}{C \vdash \mathbf{drop} : t \to \epsilon} \quad \frac{}{C \vdash \mathbf{select} : t\ t\ \mathsf{i32} \to t}$$

$$\frac{tf = t_1^n \to t_2^m \qquad C, \mathsf{label}\ (t_2^m) \vdash e^* : tf}{C \vdash \mathbf{block}\ tf\ e^*\ \mathbf{end} : tf} \quad \frac{tf = t_1^n \to t_2^m \qquad C, \mathsf{label}\ (t_1^n) \vdash e^* : tf}{C \vdash \mathbf{loop}\ tf\ e^*\ \mathbf{end} : tf}$$

$$\frac{tf = t_1^n \to t_2^m \qquad C, \mathsf{label}\ (t_2^m) \vdash e_1^* : tf \qquad C, \mathsf{label}\ (t_2^m) \vdash e_2^* : tf}{C \vdash \mathbf{if}\ tf\ e_1^*\ \mathbf{else}\ e_2^*\ \mathbf{end} : t_1^n\ \mathsf{i32} \to t_2^m}$$

$$\frac{C_{\mathsf{label}}(i) = t^*}{C \vdash \mathbf{br}\ i : t_1^*\ t^* \to t_2^*} \quad \frac{C_{\mathsf{label}}(i) = t^*}{C \vdash \mathbf{br\_if}\ i : t^*\ \mathsf{i32} \to t^*} \quad \frac{(C_{\mathsf{label}}(i) = t^*)^+}{C \vdash \mathbf{br\_table}\ i^+ : t_1^*\ t^*\ \mathsf{i32} \to t_2^*}$$

$$\frac{C_{\mathsf{label}}(|C_{\mathsf{label}}| - 1) = t^*}{C \vdash \mathbf{return} : t_1^*\ t^* \to t_2^*} \quad \frac{C_{\mathsf{func}}(i) = tf}{C \vdash \mathbf{call}\ i : tf} \quad \frac{tf = t_1^* \to t_2^* \qquad C_{\mathsf{table}} = n}{C \vdash \mathbf{call\_indirect}\ tf : t_1^*\ \mathsf{i32} \to t_2^*}$$

$$\frac{C_{\mathsf{local}}(i) = t}{C \vdash \mathbf{get\_local}\ i : \epsilon \to t} \quad \frac{C_{\mathsf{local}}(i) = t}{C \vdash \mathbf{set\_local}\ i : t \to \epsilon} \quad \frac{C_{\mathsf{local}}(i) = t}{C \vdash \mathbf{tee\_local}\ i : t \to t} \quad \frac{C_{\mathsf{global}}(i) = \mathsf{mut}^?\ t}{C \vdash \mathbf{get\_global}\ i : \epsilon \to t} \quad \frac{C_{\mathsf{global}}(i) = \mathsf{mut}\ t}{C \vdash \mathbf{set\_global}\ i : t \to \epsilon}$$

$$\frac{C_{\mathsf{memory}} = n \qquad 2^a \leq (|tp| <)^? |t| \qquad (tp\_sz)^? = \epsilon \vee t = \mathsf{i}m}{C \vdash t.\mathbf{load}\ (tp\_sz)^?\ a\ o : \mathsf{i32} \to t} \quad \frac{C_{\mathsf{memory}} = n \qquad 2^a \leq (|tp| <)^? |t| \qquad tp^? = \epsilon \vee t = \mathsf{i}m}{C \vdash t.\mathbf{store}\ tp^?\ a\ o : \mathsf{i32}\ t \to \epsilon}$$

$$\frac{C_{\mathsf{memory}} = n}{C \vdash \mathbf{current\_memory} : \epsilon \to \mathsf{i32}} \quad \frac{C_{\mathsf{memory}} = n}{C \vdash \mathbf{grow\_memory} : \mathsf{i32} \to \mathsf{i32}}$$

$$\frac{}{C \vdash \epsilon : \epsilon \to \epsilon} \quad \frac{C \vdash e_1^* : t_1^* \to t_2^* \qquad C \vdash e_2 : t_2^* \to t_3^*}{C \vdash e_1^*\ e_2 : t_1^* \to t_3^*} \quad \frac{C \vdash e^* : t_1^* \to t_2^*}{C \vdash e^* : t^*\ t_1^* \to t^*\ t_2^*}$$

Typing Modules

$$\frac{tf = t_1^* \to t_2^* \qquad C, \mathsf{local}\ t_1^*\ t^*, \mathsf{label}\ (t_2^*) \vdash e^* : \epsilon \to t_2^*}{C \vdash ex^*\ \mathbf{func}\ tf\ \mathbf{local}\ t^*\ e^* : ex^*\ tf} \quad \frac{tg = \mathsf{mut}^?\ t \qquad C \vdash e^* : \epsilon \to t \qquad ex^* = \epsilon \vee tg = t}{C \vdash ex^*\ \mathbf{global}\ tg\ e^* : ex^*\ tg}$$

$$\frac{(C_{\mathsf{func}}(i) = tf)^n}{C \vdash ex^*\ \mathbf{table}\ n\ i^n : ex^*\ n} \quad \frac{}{C \vdash ex^*\ \mathbf{memory}\ n : ex^*\ n}$$

$$\frac{}{C \vdash ex^*\ \mathbf{func}\ tf\ im : ex^*\ tf} \quad \frac{tg = t}{C \vdash ex^*\ \mathbf{global}\ tg\ im : ex^*\ tg} \quad \frac{}{C \vdash ex^*\ \mathbf{table}\ n\ im : ex^*\ n} \quad \frac{}{C \vdash ex^*\ \mathbf{memory}\ n\ im : ex^*\ n}$$

$$\frac{\begin{array}{c}(C \vdash f : ex_{\mathrm{f}}^*\ tf)^* \qquad (C_i \vdash glob_i : ex_{\mathrm{g}}^*\ tg_i)_i^* \qquad (C \vdash tab : ex_{\mathrm{t}}^*\ n)^? \qquad (C \vdash mem : ex_{\mathrm{m}}^*\ n)^? \\ (C_i = \{\mathsf{global}\ tg^{i-1}\})_i^* \qquad C = \{\mathsf{func}\ tf^*, \mathsf{global}\ tg^*, \mathsf{table}\ n^?, \mathsf{memory}\ n^?\} \qquad ex_{\mathrm{f}}^{**}\ ex_{\mathrm{g}}^{**}\ ex_{\mathrm{t}}^{*?}\ ex_{\mathrm{m}}^{*?}\ \text{distinct}\end{array}}{\vdash \mathbf{module}\ f^*\ glob^*\ tab^?\ mem^?}$$

Figure 1. Typing rules

[PLDI 2017]

mechanisation

Ocaml reference interpreter [myself / WG]

Isabelle [Conrad Watt, Cambridge]

Coq [Dave Swasey, MPI]

K (ongoing) [Everett Hildenbrandt, Illinois]

formalisation summary

off-the-shelf formal techniques

precise, concise, and comprehensive

provably correct and machine-verified

feedback loop let to clean and simple design

standard

includes complete formalisation!

spec Cobol is still included,
but just a redundant text rendering

webassembly.github.io/spec

embedding

standard is layered

js & web cruft confined in embedder specs

other embeddings are equally possible!

other embedders

mobile platforms (e.g., Android)

content sandboxing (e.g., Fastly)

decentralised cloud / “smart contracts” (e.g., Ethereum)

embedded devices

standalone implementations

next up

more performance
more languages
more platforms
more features
more tools

road map

v1 (shipped): focus on low-level languages

v2 (this/next year): high-level languages

v3 (later): “dynamic” languages

future features

threads

SIMD

tail calls

exception handling

stack switching (effect handlers?)

reference types

garbage collection

proposal process

must include spec text

must include formalisation!

must include Ocaml reference interpreter extension

must include comprehensive test suite

must be implemented in 2 production engines

Summary

Efficient, safe, sandboxed, universal

Open, public standard process

Formal rigour and machine verification in the mainstream

Let to a clean and simple design

Don’t let the name fool you!

webassembly.github.io/spec

webassembly.org

Thank you.