WebAssembly

Contents

- Overview
- Semantic Phases
- Future Works

Overview

가상 명령어 집합 구조 – Virtual Instruction Set Architecture(V-ISA)

C++	Binary	Text

`int factorial(int n) {`
`  if (n == 0)`
`  return 1;`
`  else`
`  return n * factorial(n-1);`
`}`
	`20 00`
	`42 00`
	`51`
	`04 7e`
	`42 01`
	`05`
	`20 00`
	`20 00`
	`42 01`
	`7d`
	`10 00`
	`7e`
	`0b`
		`get_local 0`
		`i64.const 0`
		`i64.eq`
		`if i64`
		`  i64.const 1`
		`else`
		`  get_local 0`
		`  get_local 0`
		`  i64.const 1`
		`  i64.sub`
		`  call 0`
		`  i64.mul`
		`end`

Overview

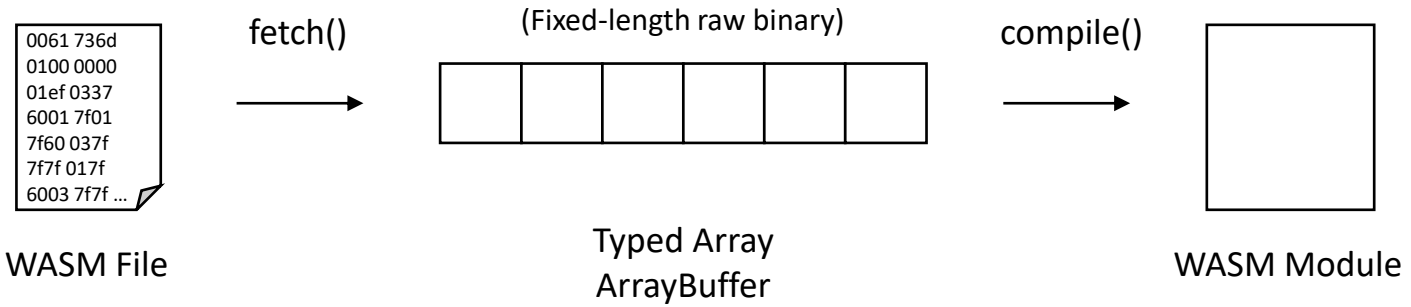
- Uses only four variable types
  - i32, i64, f32, f64
  - i32 and i64 are not inherently signed or unsigned
- Can be used with Web APIs
  - DOM, IndexedDB, Web Workers, etc…
- Not a substitute for JavaScript!

Semantic Phases

- Decoding -> Validation -> Execution

- Execution is divided into two phases:
  - Instantiation vs. Invocation

Decoding


0061 736d
0100 0000
01ef 0337
6001 7f01
7f60 037f
7f7f 017f
6003 7f7f ...
fetch()
(Fixed-length raw binary)
compile()
WASM File
Typed Array
ArrayBuffer
WASM Module

Validation / Execution

(value types) $t ::= \mathsf{i32} \mid \mathsf{i64} \mid \mathsf{f32} \mid \mathsf{f64}$
(packed types) $tp ::= \mathsf{i8} \mid \mathsf{i16} \mid \mathsf{i32}$
(function types) $tf ::= t^\star \to t^\star$
(global types) $tg ::= \mathsf{mut}^? \; t$

$unop_{\mathsf{i}N} ::= \mathsf{clz} \mid \mathsf{ctz} \mid \mathsf{popcnt}$
$unop_{\mathsf{f}N} ::= \mathsf{neg} \mid \mathsf{abs} \mid \mathsf{ceil} \mid \mathsf{floor} \mid \mathsf{trunc} \mid \mathsf{nearest} \mid \mathsf{sqrt}$
$binop_{\mathsf{i}N} ::= \mathsf{add} \mid \mathsf{sub} \mid \mathsf{mul} \mid \mathsf{div}\_sx \mid \mathsf{rem}\_sx \mid$
$\mathsf{and} \mid \mathsf{or} \mid \mathsf{xor} \mid \mathsf{shl} \mid \mathsf{shr}\_sx \mid \mathsf{rotl} \mid \mathsf{rotr}$
$binop_{\mathsf{f}N} ::= \mathsf{add} \mid \mathsf{sub} \mid \mathsf{mul} \mid \mathsf{div} \mid \mathsf{min} \mid \mathsf{max} \mid \mathsf{copysign}$
$testop_{\mathsf{i}N} ::= \mathsf{eqz}$
$relop_{\mathsf{i}N} ::= \mathsf{eq} \mid \mathsf{ne} \mid \mathsf{lt}\_sx \mid \mathsf{gt}\_sx \mid \mathsf{le}\_sx \mid \mathsf{ge}\_sx$
$relop_{\mathsf{f}N} ::= \mathsf{eq} \mid \mathsf{ne} \mid \mathsf{lt} \mid \mathsf{gt} \mid \mathsf{le} \mid \mathsf{ge}$
$cvtop ::= \mathsf{convert} \mid \mathsf{reinterpret}$
$sx ::= \mathsf{s} \mid \mathsf{u}$

(instructions) $e ::= \mathsf{unreachable} \mid \mathsf{nop} \mid \mathsf{drop} \mid \mathsf{select} \mid$
$\mathsf{block}\ tf\ e^\star\ \mathsf{end} \mid \mathsf{loop}\ tf\ e^\star\ \mathsf{end} \mid \mathsf{if}\ tf\ e^\star\ \mathsf{else}\ e^\star\ \mathsf{end} \mid$
$\mathsf{br}\ i \mid \mathsf{br\_if}\ i \mid \mathsf{br\_table}\ i^+ \mid \mathsf{return} \mid \mathsf{call}\ i \mid \mathsf{call\_indirect}\ tf \mid$
$\mathsf{get\_local}\ i \mid \mathsf{set\_local}\ i \mid \mathsf{tee\_local}\ i \mid \mathsf{get\_global}\ i \mid$
$\mathsf{set\_global}\ i \mid t.\mathsf{load}\ (tp\_sx)^?\ a\ o \mid t.\mathsf{store}\ tp^?\ a\ o \mid$
$\mathsf{current\_memory} \mid \mathsf{grow\_memory} \mid t.\mathsf{const}\ c \mid$
$t.unop_t \mid t.binop_t \mid t.testop_t \mid t.relop_t \mid t.cvtop\ t\_sx^?$

(functions) $f ::= ex^\star\ \mathsf{func}\ tf\ \mathsf{local}\ t^\star\ e^\star \mid ex^\star\ \mathsf{func}\ tf\ im$
(globals) $glob ::= ex^\star\ \mathsf{global}\ tg\ e^\star \mid ex^\star\ \mathsf{global}\ tg\ im$
(tables) $tab ::= ex^\star\ \mathsf{table}\ n\ i^\star \mid ex^\star\ \mathsf{table}\ n\ im$
(memories) $mem ::= ex^\star\ \mathsf{memory}\ n \mid ex^\star\ \mathsf{memory}\ n\ im$
(imports) $im ::= \mathsf{import}\ \text{“}name\text{”}\ \text{“}name\text{”}$
(exports) $ex ::= \mathsf{export}\ \text{“}name\text{”}$
(modules) $m ::= \mathsf{module}\ f^\star\ glob^\star\ tab^?\ mem^?$

Figure 1. WebAssembly abstract syntax

Validation / Execution

Functions *func* are classified by *function types* of the form $[t_1^*] \to [t_2^?]$.

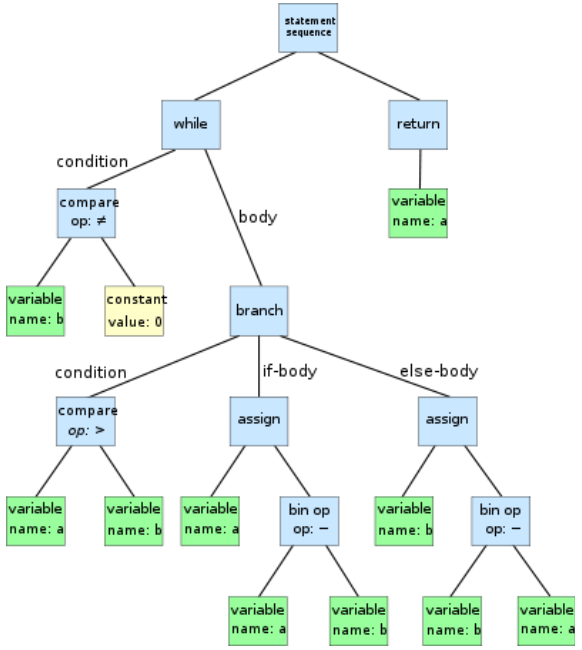
$\{\text{type } x, \text{locals } t^*, \text{body } expr\}$

- The type $C.\text{types}[x]$ must be defined in the context.
- Let $[t_1^*] \to [t_2^?]$ be the *function type* $C.\text{types}[x]$.
- Let C' be the same *context* as C, but with:
  - locals set to the sequence of *value types* $t_1^*\ t^*$, concatenating parameters and locals,
  - labels set to the singular sequence containing only *result type* $[t_2^?]$.
  - return set to the *result type* $[t_2^?]$.
- Under the context C', the expression $expr$ must be valid with type $t_2^?$.
- Then the function definition is valid with type $[t_1^*] \to [t_2^?]$.

$$\frac{C.\text{types}[x] = [t_1^*] \to [t_2^?] \qquad C, \text{locals } t_1^*\ t^*, \text{labels } [t_2^?], \text{return } [t_2^?] \vdash expr : [t_2^?]}{C \vdash \{\text{type } x, \text{locals } t^*, \text{body } expr\} : [t_1^*] \to [t_2^?]}$$

Module

- AST
  - Abstract Syntax Tree
- S-expression
  - Notation for nested tree-structured data
- Unit of deployment / loading / compilation



Example in Common Lisp:

```
(defun factorial (x)
   (if (zerop x)
       1
       (* x (factorial (- x 1)))))
```

Module

$$
\begin{array}{lll}
module & ::= & \{ \quad \text{types } vec(functype), \\
 & & \quad \text{funcs } vec(func), \\
 & & \quad \text{tables } vec(table), \\
 & & \quad \text{mems } vec(mem), \\
 & & \quad \text{globals } vec(global), \\
 & & \quad \text{elem } vec(elem), \\
 & & \quad \text{data } vec(data), \\
 & & \quad \text{start } start^{?}, \\
 & & \quad \text{imports } vec(import), \\
 & & \quad \text{exports } vec(export) \quad \}
\end{array}
$$

- Can declare…
  - init logic or start function
  - imports and exports

imports / exports

- import/exportable definitions(defined and referenced with indices):
  - Functions
  - Tables
  - Memories
  - Globals

Validation / Execution

- Instantiation
  - Complete with its own state and execution stack
  - Executes given definitions for all its imports
  - = Initialises globals, memories and tables
  - Returns the instances of the module’s exports

- Invocation
  - Executes the exported functions and returns its results

Future Works

- Linear Memory
- Table
- Implementation without emscripten