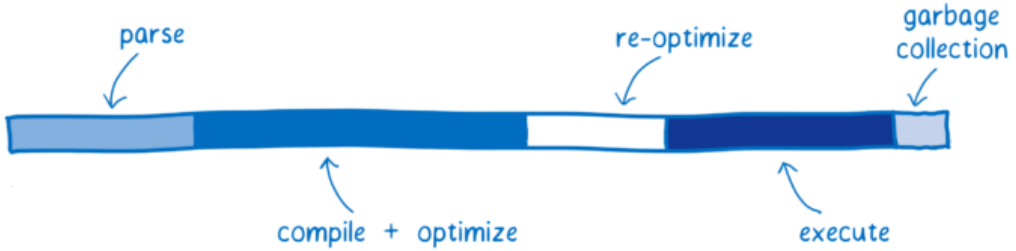
WEBASSEMBLY

Μεταγλωττίζοντας για τον browser με WebAssembly

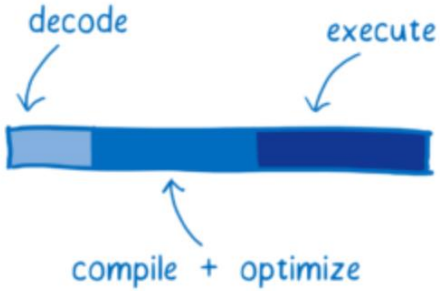
Αλέξανδρος Μόσχος

Το πρόβλημα

Πόσο γρήγορη είναι η js;



Οι σύγχρονοι browser έχουν βελτιώσει πάρα πολύ την ταχύτητα της JavaScript. Όμως υπάρχει αρκετός χώρος για βελτίωση.
Πιο γρήγορος clientside κώδικας =>
1. Περισσότερες εφαρμογές για το web (Gaming μέσα στον browser με client-side rendering)
2. Λιγότεροι χαμένοι κύκλοι ρολογιού του επεξεργαστή => Καλύτερη εξοικονόμηση μπαταρίας (ιδιαίτερα σημαντικό σε mobile συσκευές)
Εαν μπορέσουμε να δημιουργήσουμε ενα νέο πρότυπο θα θέλαμε ο χρόνος εκτέλεσης στον browser να μοιάζει κάπως έτσι



Plus operator in JavaScript

```
function add(a, b) {
    return a + b;
}
```

Ο τελεστής ‘+’ σε JavaScript που (λογικά) θα θέλαμε να αθροίζει δυο αριθμούς έχει αρκετά περίπλοκη σημασιολογία (https://www.ecma-international.org/ecma-262/5.1/#sec-11.6.1)

11.6.1 The Addition operator (+)
The addition operator either performs string concatenation or numeric addition.

The production AdditiveExpression : AdditiveExpression + MultiplicativeExpression is evaluated as follows:

Let lref be the result of evaluating AdditiveExpression.
Let lval be GetValue(lref).
Let rref be the result of evaluating MultiplicativeExpression.
Let rval be GetValue(rref).
Let lprim be ToPrimitive(lval).
Let rprim be ToPrimitive(rval).
If Type(lprim) is String or Type(rprim) is String, then
Return the String that is the result of concatenating ToString(lprim) followed by ToString(rprim)
Return the result of applying the addition operation to ToNumber(lprim) and ToNumber(rprim). See the Note below 11.6.3.

Plus operator in JavaScript(2)

```
function add(a, b) {
    return a + b;
}
```

Ο κώδικας που εκτελεί ο browser για να εκτελέσει το παραπάνω JavaScript snippet είναι αρκετά μεγάλος(firefox/js/src/vm/Interpreter.cpp)

Plus operator in JavaScript(3)

```
function add(a, b) {
    return a + b;
}
```

Ο κώδικας που εκτελεί ο browser για να εκτελέσει το παραπάνω JavaScript snippet είναι αρκετά μεγάλος(firefox/js/src/vm/Interpreter.cpp)

```
if (lhs.isInt32() && rhs.isInt32()) {
    int32_t l = lhs.toInt32(), r = rhs.toInt32();
    int32_t t;
    if (MOZ_LIKELY(SafeAdd(l, r, &t))) {
        res.setInt32(t);
        return true;
    }
}
if (!ToPrimitive(cx, lhs)) return false;
if (!ToPrimitive(cx, rhs)) return false;
bool lIsString, rIsString;
if ((lIsString = lhs.isString()) | (rIsString = rhs.isString())) {
    JSString* lstr;
    if (lIsString) {
        lstr = lhs.toString();
    } else {
        lstr = ToString<CanGC>(cx, lhs);
        if (!lstr) return false;
    }
    JSString* rstr;
    if (rIsString) {
        rstr = rhs.toString();
    } else {
        lhs.setString(lstr);
        rstr = ToString<CanGC>(cx, rhs);
        if (!rstr) return false;
        lstr = lhs.toString();
    }
    JSString* str = ConcatStrings<NoGC>(cx, lstr, rstr);
    if (!str) {
        RootedString nlstr(cx, lstr), nrstr(cx, rstr);
        str = ConcatStrings<CanGC>(cx, nlstr, nrstr);
        if (!str) return false;
    }
    res.setString(str);
} else {
    double l, r;
    if (!ToNumber(cx, lhs, &l) || !ToNumber(cx, rhs, &r)) return false;
    res.setNumber(l + r);
}
```

Παλιές προσεγγίσεις - Plugins

Παλια για να γράψουμε γρήγορο κώδικα στον browser γράφαμε plugins.

- Τα plugins δεν είναι πάντοτε ασφαλή
- Δεν υπάρχει καλή ενσωμάτωση με τα Web APIs
- Πολύ σπάνια τρέχουν σε mobile

Συγκεκριμένα στην σελίδα του MDN για τα plugins το παρακάτω βρίσκεται στην κορυφή:

> **Important**: plugins are a legacy technology that are a security and performance problem for Firefox (and other browser) users. They may not be supported in the future. New content should not be written using Flash or any other plugin technology.

Παλιές προσεγγίσεις – asm.js

- Το asm.js είναι ενα subset της JavaScript που έχει σκοπό να βελτιώσει την επίδοση δίνοντας συγκεκριμένα annotations στον browser για να τρέξει πιο γρήγορα την JavaScript. Αφου είναι υποσύνολο της JavaScript είναι ήδη sandboxed.
- Είναι backwards compatible και λειτουργεί σε καθε browser
- Είναι «ενδιάμεση» γλώσσα δηλαδή ένα πρόγραμμα C++ μπορεί με κατάλληλο εργαλείο να γίνει cross-compiled με target την asm.js και θα παραχθεί ένα αρχείο JavaScript το οποίο μπορούμε να το στείλουμε στον client για να το εκτελέσει στον browser.

```
function f(i) {
    i = i|0;
    return (i + 1)|0;
}
```

- Μοιάζει με χακιά
- Η βελτίωση της ταχύτητας δεν είναι και τόσο μεγάλη
- Οχι πολύ καλα debug tools για τον front end developer

Τι θέλουμε

- Αποδοτική εκτέλεση στον browser
- Ασφάλεια
- Επικοινωνία με την JavaScript
- Καλά εργαλεία για debugging
- Να τρέχει σε όλους τους browser desktop και mobile
- Να μικρύνουμε τα δεδομένα που αποστέλλονται στον client
- Ήδη υπάρχουσες native εφαρμογές να μπορούν με μικρές αλλαγές να τρέξουν στο web

Τι θέλουμε

- Αποδοτική εκτέλεση στον browser
- Ασφάλεια
- Επικοινωνία με την JavaScript
- Καλά εργαλεία για debugging
- Να τρέχει σε όλους τους browser desktop και mobile
- Να μικρύνουμε τα δεδομένα που αποστέλλονται στον client
- Ήδη υπάρχουσες native εφαρμογές να μπορούν με μικρές αλλαγές να τρέξουν στο web

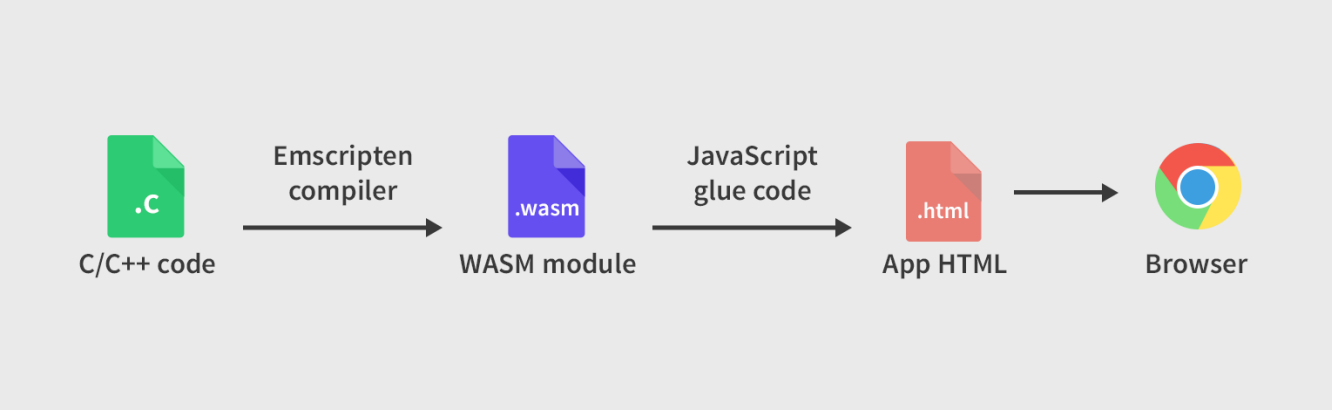
Λύση: WebAssembly

What is WebAssembly

Η WebAssembly είναι standard που καθορίζει ενα byte code format που υποστηρίζεται πλέον από όλους τους σύγχρονους browser απο την Mozilla, την Google και την Microsoft.

Πως τρέχει στον client

- Αντί να στέλνουμε 1 αρχείο html, 1 αρχείο css και 1 αρχείο js, στέλνουμε και 1 αρχείο wasm.
- Πρώτα εκτελείται η JavaScript και ο κώδικας js μπορεί να καλέσει κάποια συνάρτηση WebAssembly που γράψαμε εμείς.
- Χρησιμοποιεί το ήδη υπάρχον VM της JavaScript μεσα στον browser. Οπότε μπορουν να εφαρμοστούν οι βελτιστοποιήσεις που υλοποιεί ο browser.



Χρησιμότητα

Υπάρχουν δύο αναπαράστασεις της WebAssembly

1. Ενα μικρό σε μέγεθος binary format με σκοπό να αποσταλλεί και να φορτωθεί γρήγορα απο τον client
2. Ενα text format που είναι εντελώς ισοδύναμο

Βασικές χρήσεις:

1. Compile target για το Web
2. Χρησιμο εργαλείο για την δημιουργία βιβλιοθηκών JavaScript

Πως την χρησιμοποιούμε;

Γράφουμε ενα πρόγραμμα (χωρις αυτόματη διαχείριση μνήμης) και ο compiler μας το κάνει cross compile σε WebAssembly. Μέχρι στιγμής υποστηρίζονται C/C++ και Rust

High level goals-Binary code

- Μικρό σε μέγεθος format και ειδικότερα μικρότερο απο gzipped asm.js
- Σε ένα γραμμικό πέρασμα να μπορεί να γίνει compilation και verification του κώδικα. Ο κώδικας τρέχει μεσα σε sandbox οπότε δεν μπορούμε να επιτρέπουμε προσβάσεις στην μνήμη σε οποιαδήποτε διεύθυνση. Δεν είναι απαραίτητο να γίνει verification πριν την εκτέλεση
- Να είναι εύκολα επεκτάσιμο.

WebAssembly Memory Model

- Όταν εκτελούνται προγράμματα C έχουμε stack και heap. Ενας δείκτης μπορεί να δείχνει οπουδήποτε.
- Για λόγους ασφαλείας η WebAssembly έχει διαφορετικό μοντέλο εκτέλεσης απο την C. Αντί για το heap έχουμε την γραμμική μνήμη και έχουμε το execution stack. Είναι πολυ σημαντικό για λόγους ασφαλείας η execution stack να είναι isolated. Θα μπορούσαμε να κάνουμε runtime έλεγχο για να βεβαιωθούμε οτι οι δείκτες δείχνουν εντός της γραμμικής μνήμης
- Επίσης οι συναρτήσεις δεν καλούνται μέσω δεικτών αλλα μέσω ενός ακεραίου offset που χρησιμοποιείται για να δεικτοδοτήσει το function table.

0 mem_size

Function Table

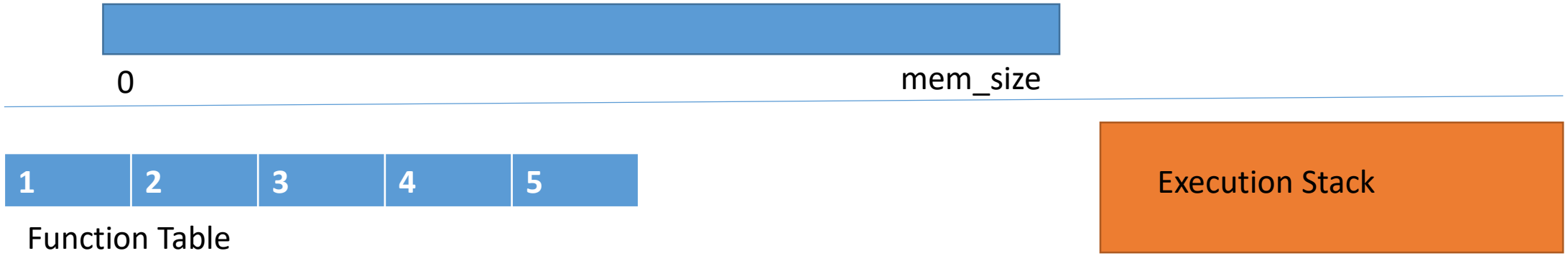
Execution Stack

WebAssembly Local Variables

- Ο παρακάτω κώδικας, σε μια μη προσεκτική υλοποίηση θα δημιουργούσε πρόβλημα

```
int foo(){
    int local;
    bar(&local);
}
```

- Για αυτόν τον λόγο η execution stack δεν περιέχει ακριβώς εγγραφές δραστηριοποιήσης των συναρτήσεων, αλλά μια συντηρητική μορφή τους.
- Κάθε τοπική μεταβλητή στην οποία δεν χρησιμοποιείται & τοποθετείται στην εγγραφη δραστηριοποίησης και οι υπόλοιπες τοπικές τοποθετούνται σε ειδικό χώρο στην γραμμική μνήμη(παλι στοίβα για να υλοποιείται σωστά η αναδρομή).



WebAssembly Semantics

Ο κώδικας WebAssembly μπορει να θεωρηθεί ως structured stack machine.

Οι εντολές χωρίζονται σε 2 είδη:

- Simple Instructions: Κάνουν pop τα ορίσματα τους απο την στοίβα και push το αποτέλεσμα στην στοίβα(και ο program counter προχωρά στην επόμενη εντολή)
- Control Instructions: Κάνουν pop τα ορισματά τους απο την στοίβα και αλλάζουν κατάλληλα τον program counter(και παραλλήλα μπορούν να κάνουν push ενα αποτέλεσμα στην στοίβα)

Κάποιες εντολές μπορούν να οδηγηθούν σε trap υπο ορισμένες συνθήκες. Όταν προκληθεί trap έχουμε throw μιας JavaScript exception

WebAssembly Text Format

Ενα WebAssembly module είναι ένα μεγάλο S-expression(LISP). Τα S-expressions είναι ένα text format για την αναπαράσταση αφηρημένου συντακτικού δέντρου

```
type ::i32 | i64 | f32 | f64 | void

func ::(func <signature> <locals> <body>)

instr ::
      <expr>
      <op>
      block <name>? <block_sig> <instr>* end <name>?
      loop <name>? <block_sig> <instr>* end <name
      if <name>? <block_sig> <instr>* end <name>?
      if <name>? <block_sig> <instr>* else <name>? <instr>* end <name>?

op ::
      nop
      br
      br_if
      get_local
      set_local
      call
      return
      <type>.<binop>
      <type>.<load|store>
      …

module ::
      (module <name>? <typedef>* <func>* <import>* <export>* <table>? <memory>? <global>* <elem>* <data>* <start>?) …
```

WebAssembly Examples

Παράδειγμα αθροίσματος δύο double αριθμών

```
(func (;19;) (type 6) (param f64 f64) (result f64)
  get_local 0
  get_local 1
  f64.add)
```

Παράδειγμα mandelbrot

C

```
int colors[N * N * 4];
void main () {
  for (int y = 0; y < N; y++) {
    for (int x = 0; x < N; x++) {
      vec2 coordinate = { .x = (float) x, .y =
(float) y };
      vec2 cartesianPosition =
fragCoordToXY(coordinate);
      float color = mandelbrot(cartesianPosition);
      int position = (y * N + x) * 4;
      colors[position + 0] = color * 20;
      colors[position + 1] = color * 20;
      colors[position + 2] = color * 20;
      colors[position + 3] = 255;
    }
  }
}
```

JavaScript

```
WebAssembly.instantiate(wasmCode,{}).then(({instance
}) => {
  var memory = instance.exports.memory
  var i32 = new
Uint32Array(instance.exports.memory.buffer);
  instance.exports.main()
  var colors = i32.slice(offset, offset + N * N * 4)
  var ctx = canvas.getContext('2d');
  var imageData = ctx.getImageData(0, 0, N, N);
  var data = imageData.data;
  for (var i = 0; i < colors.length; i++) {
    data[i] = colors[i]
  }
  ctx.putImageData(imageData, 0, 0);
  lib.showCanvas()
});
```

WebAssembly Examples(2)

Παράδειγμα αντίστροφης τετραγωνικής ρίζας

```
float Q_rsqrt(float number) {
    long i;
    float x2, y;
    const float threehalfs = 1.5F;
    x2 = number * 0.5F;
    y = number;
    i = *(long *) &y;
    i = 0x5f3759df - (i >> 1); //WTF?
    y = *(float *) &i;
    y = y * (threehalfs-(x2 * y * y));
    y = y * (threehalfs-(x2 * y * y));
    return y;
}
```

```
(func $func0 (param $var0 f32) (result f32)
(local $var1 f32)
f32.const 1.5
get_local $var0
f32.const 0.5
f32.mul
tee_local $var1
i32.const 1597463007
get_local $var0
i32.reinterpret/f32
i32.const 1
i32.shr_s
i32.sub
f32.reinterpret/i32
tee_local $var0
f32.mul
get_local $var0
f32.mul
f32.sub
get_local $var0
f32.mul
tee_local $var0
f32.const 1.5
get_local $var0
get_local $var1
get_local $var0
f32.mul
f32.mul
f32.sub
f32.mul
)
```

```
wasm-function[0]:
sub rsp, 8
movss xmm1, dword ptr [rip + 0x844]
movaps xmm3, xmm1
movss xmm1, dword ptr [rip + 0x83d]
mulss xmm1, xmm0
movd eax, xmm0
sar eax, 1
mov ecx, 0x5f3759df
sub ecx, eax
movd xmm0, ecx
movaps xmm4, xmm1
mulss xmm4, xmm0
mulss xmm4, xmm0
movaps xmm2, xmm3
subss xmm2, xmm4
mulss xmm2, xmm0
mulss xmm1, xmm2
mulss xmm1, xmm2
movaps xmm0, xmm3
subss xmm0, xmm1
mulss xmm0, xmm2
nop
add rsp, 8
ret
```

Συχνές ερωτήσεις

Είναι assembly για κάποιον επεξεργαστή;

Όχι, είναι ενδιάμεση αναπαράσταση που μοιάζει με LLVM IR και .NET bytecode. Είναι το instruction set ενός «φανταστικού» CPU που τρέχει μεσα στον browser.

Είναι Java Applet / Plugin 2.0 / ActiveX;

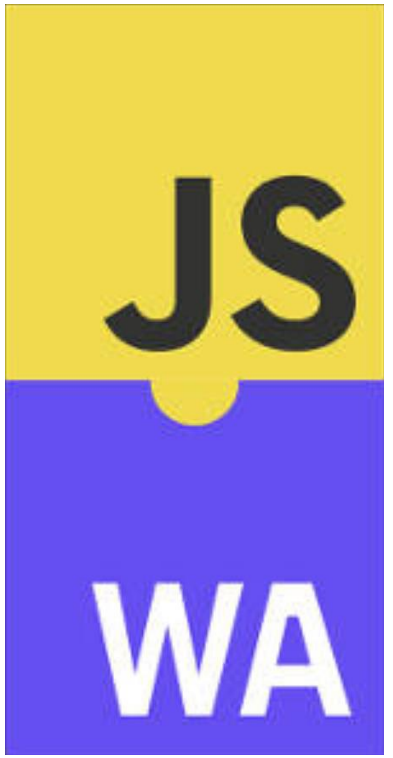
Όχι ο browser δεν χρειάζεται να σηκώσει άλλον VM για την εκτέλεση κώδικα WebAssembly. Δεν εκτελείται ποτέ native κώδικας οπότε αν το browser sandbox είναι ασφαλές τότε είναι και ο client.

Μπορώ να κάνω system calls απο WebAssembly?

Όχι αν μπορούσαν να εκτελεστούν system calls θα είχε παραβιαστεί η ασφάλεια του browser. Ωστόσο, σε ορισμένες περιπτώσεις οι κατασκευαστές compiler για κάποια API κάνουν την αντιστοίχιση σε API του browser(π.χ. Γράφω κώδικα για sockets και τελικά στον client εκτελείται κώδικας που κάνει χρήση WebSockets).

Θα αντικαταστήσει την JS;

- Οχι, το πρότυπο WebAssembly μέχρι στιγμής δεν υποστηρίζει garbage collection.
- Ο σχεδιασμός της wasm έγινε με τέτοιο τρόπο έτσι ώστε ο προγραμματιστής να κάνει DOM manipulation με js και compute heavy tasks με wasm.
- Η μη υποστήριξη garbage collection στο πρότυπο της WebAssembly δεν σημαίνει ότι δεν μπορούν να υποστηριχτούν γλώσσες με garbage collection. Θα μπορούσε το WebAssembly module να υλοποιεί το δικό του garbage collector(βλ. Kotlin Native)
- Ωστόσο μία γλώσσα που μοιάζει στην js θα μπορούσε να εκμεταλλευτεί την επίδοση της wasm αρκεί να μην είχε garbage collector(TypeScript ??). Η Turboscript είναι μια «διαλεκτος» JavaScript που κάνει explicit memory allocation και deallocation και κάνει compile σε wasm



Compilers και εργαλεία

- Ο βασικός compiler που δέχεται ως είσοδο C/C++ και παράγει WebAssembly είναι ο Emscripten. Κάνει χρήση ενός προγράμματος που λέγεται Binaryen που μετατρέπει asm.js κώδικα σε κώδικα wasm.
- Η ομάδα πίσω απο το emscripten έχει ήδη ξεκινήσει την δημιουργία αυτόνομου compiler που θα λαμβάνει κώδικα C/C++ και θα παράγει wasm χωρίς το ενδιάμεσο βήμα παραγωγής asm.js
- Η ομάδα πίσω απο τον clang/LLVM έχει ξεκινήσει μαζί με μηχανικούς απο τους 4 μεγαλύτερους browser να παράγει ένα νεο backend για το LLVM που θα παράγει κώδικα WebAssembly.
- Η «καλή» υποστήριξη C/C++/Rust φαίνεται να αργεί λίγο αφού οι γλώσσες είναι περίπλοκες.
- Η TurboScript υποστηρίζει παραγωγή κώδικα wasm και έχει ήδη έτοιμο compiler.

Performance

- Κατα μέσο όρο σε ένα video animation benchmark η wasm έχει 10x καλύτερη ταχύτητα απο την Javascript σε Windows Android και Linux.
- Αλλα εκτος απο benchmarks στην καθημερινή χρήση το wasm bytecode είναι μικρότερο απο asm.js και είναι πιο εύκολο να γίνει parse.
- Στο μέλλον με τις επόμενες εκδόσεις wasm θα προστεθούν
  - Threads
  - SIMD
  - Garbage collection(Proposed)
  - Tail call optimization
  - Direct DOM API(Proposed)

Περισσότερες πληροφορίες

- Τα έγγραφα σχεδιασμού της WebAssembly βρίσκονται στο GitHub. https://github.com/WebAssembly/design
- Ο σχεδιασμός της WebAssembly στο παραπάνω repository περιλαμβάνει ανάλυση απαιτήσεων, καθορισμό του Minimum viable product, usecase analysis.
- Η reference υλοποίηση του interpreter που εκτελεί wasm bytecode έχει γραφτεί σε OCaml https://github.com/WebAssembly/spec/tree/master/interpreter
- Η wasm υποστηρίζεται απο όλους τους browser σήμερα, αλλα πρέπει να υλοποιηθούν καλύτερα developer tools.
- Για μεταγλώττιση κώδικα C σε wasm και εκτέλεση online υπαρχεί το https://wasdk.github.io/WasmFiddle/
- Για μετατροπη κώδικα Rust/C/C++ σε wasm text format και εφαρμογή optimizations (και παραγωγή x86 κώδικα) https://mbebenita.github.io/WasmExplorer/

Thank You!