WIKIPEDIA

WebAssembly

WebAssembly (often shortened to **Wasm**) is an open standard that defines a portable binary-code format for executable programs, and a corresponding textual assembly language, as well as interfaces for facilitating interactions between such programs and their host environment.[1][2][3][4] The main goal of WebAssembly is to enable high-performance applications on web pages, but the format is designed to be executed and integrated in other environments as well.[5][6]

WebAssembly became a World Wide Web Consortium recommendation on 5 December 2019[7] and, alongside HTML, CSS, and JavaScript, is the fourth language to run natively in browsers.[8] In order to use Wasm in browsers, users may use Emscripten SDK to compile C++ (or any other LLVM-supported language such as D or Rust) source code into a binary file which runs in the same sandbox as regular JavaScript code.[note 1] Emscripten provides bindings for several commonly used environment interfaces like WebGL. There is no direct Document Object Model (DOM) access; however, it is possible to create proxy functions for this, for example through stdweb,[13] web_sys,[14] and js_sys[15] when using Rust language.

The World Wide Web Consortium (W3C) maintains the standard with contributions from Mozilla, Red Hat, Microsoft, Google, and Apple.[16][17]

WebAssembly

Paradigm	Imperative, structured, expression-oriented
Designed by	W3C
Developer	W3C Mozilla Microsoft Google Apple
First appeared	March 2017
Typing discipline	Static
License	Apache License 2.0
Filename extensions	.wat .wasm
Website	webassembly.org (https://webassembly.org/)

Influenced by

asm.js · PNaCl

Contents

History

WebAssembly was first announced in 2015,[18] and the first demonstration was executing Unity's *Angry Bots* in Firefox,[19] Google Chrome,[20] and Microsoft Edge.[21] The precursor technologies were asm.js from Mozilla and Google Native Client,[22][23] and the initial implementation was based on the feature set of asm.js.[24] The asm.js technology already provides near-native code execution speeds[25] and can be considered a viable alternative for browsers that don't support WebAssembly or have it disabled for security reasons.

In March 2017, the design of the minimum viable product (MVP) was declared to be finished and the preview phase ended.[26] In late September 2017, Safari 11 was released with support. In February 2018, the WebAssembly Working Group published three public working drafts for the Core Specification, JavaScript Interface, and Web API.[27][28][29][30]

Support

In November 2017, Mozilla declared support "in all major browsers"[31] (by now all major on mobile and desktop), after WebAssembly was enabled by default in Edge 16.[32] The support includes mobile web browsers for iOS and Android. As of May 2020, 91.66% of installed browsers (91.65% of desktop browsers and 93.32% of mobile browsers) support WebAssembly.[33] But for older browsers, Wasm can be compiled into asm.js by a JavaScript polyfill.[34]

Because WebAssembly executables are precompiled, it is possible to use a variety of programming languages to make them.[35] This is achieved either through direct compilation to Wasm, or through implementation of the corresponding virtual machines in Wasm. There have been around 40 programming languages reported to support Wasm as a compilation target.[36]

Emscripten compiles C and C++ to Wasm[26] using the LLVM backend.[37]

As of version 8[38] a standalone Clang can compile C and C++ to Wasm.

Its initial aim is to support compilation from C and C++,[39] though support for other source languages such as Rust and .NET languages is also emerging.[40][41][36] After the MVP release, there are plans to support multithreading and garbage collection[42][43] which would make WebAssembly a compilation target for garbage-collected programming languages like C# (supported via Blazor), F# (supported via Bolero[44] with help of Blazor), Python, and even JavaScript where the browser's Just-in-time compilation speed is considered too slow. A number of other languages have some support including Java, Julia,[45][46][47] Ruby,[48] as well as Go.

Security considerations

In June 2018, a security researcher presented the possibility of using WebAssembly to circumvent browser mitigations for Spectre and Meltdown security vulnerabilities once support for threads with shared memory is added. Due to this concern, WebAssembly developers put the feature on hold.[49][50][51] However, in order to explore these future language extensions, Google Chrome added experimental support for the WebAssembly thread proposal in October 2018.[52]

WebAssembly has been criticized for allowing greater ease of hiding the evidence for malware writers, scammers and phishing attackers; WebAssembly is only present on the user's machine in its compiled form, which "[makes malware] detection difficult".[53] The speed and concealability of WebAssembly have led to its use in hidden crypto mining on the website visitor's device.[53][54][49] Coinhive, a now defunct service facilitating cryptocurrency mining in website visitors' browsers, claims their "miner uses WebAssembly and runs with about 65% of the performance of a native Miner."[49] A June 2019 study from the Technische Universität Braunschweig, analyzed the usage of WebAssembly in the Alexa top 1 million websites and found the prevalent use was for malicious crypto mining, and that malware accounted for more than half of the WebAssembly-using websites studied.[55][56]

The ability to effectively obfuscate large amounts of code can also be used to disable Ad blocking and privacy tool that prevent Web tracking like Privacy Badger

As WebAssembly only supports structured control flow, it is amenable toward security verification techniques including symbolic execution. Current efforts in this direction include the Manticore symbolic execution engine.[57]

Embedding

The general standards provide core specifications for JavaScript and Web embedding.[3]

While WebAssembly was initially designed to enable near-native code execution speed in the web browser, it has been considered valuable outside of such, in more generalized contexts.[58][59]

WASI

WebAssembly System Interface (WASI) is a simple interface (ABI and API) designed by Mozilla intended to be portable to any platform.[60] It provides POSIX features like file I/O constrained by capability-based security.[61][62] There are also a few other proposed ABI/APIs.[63][64]

WASI was influenced by CloudABI and Capsicum.

Specification

Virtual machine

Wasm code (binary or bytecode) is intended to be run on a portable virtual stack machine (VM).[65] The VM is designed to be faster to parse and execute than JavaScript and to have a compact code representation.[39]

Wasm program

A Wasm program is designed to be a separate module containing collections of various wasm-defined values and program type definitions. These are expressed in either binary or textual format (see below) that both have a common structure.[66]

Instruction set

The core standard for the binary format of a wasm program defines an instruction set architecture consisting of specific binary encodings of types of operations which are executed by the VM. It doesn't specify how exactly they must be executed by the VM however.[67] The list of instructions includes standard memory load/store instructions, numeric, parametric, control of flow instruction types and wasm-specific variable instructions.[68]

Code representation

In March 2017, the WebAssembly Community Group reached consensus on the initial (MVP) binary format, JavaScript API, and reference interpreter.[69] It defines a WebAssembly binary format (`.wasm`), which is not designed to be used by humans, as well as a human-readable WebAssembly text format (`.wat`) that resembles a cross between S-expressions and traditional assembly languages.

The table below represents three different views of the same source code input from the left, as it is converted to a Wasm intermediate representation, then to Wasm binary instructions:[70]

The same source code in C, assembly, and Wasm

C input source	Linear assembly bytecode (intermediate representation)	Wasm binary encoding (hexadecimal bytes)
`int factorial(int n) {`	`; magic number`	`00 61 73 6D 01 00 00 00`
`  if (n == 0)`	`; type for (func (param i64) (result`	`01 00 01 60 01 73 01 73 06`
`    return 1;`	`i64))`	`03 00 01 00 02`
`  else`	`; function section`	`0A 00 01`
`    return n *`	`; code section start`	`00 00`
`factorial(n-1);`	`(func (param i64) (result i64)`	`20 00`
`}`	`  local.get 0`	`50`
	`  i64.eqz`	`04 7E`
	`  if (result i64)`	`42 01`
	`      i64.const 1`	`05`
	`  else`	`20 00`
	`      local.get 0`	`20 00`
	`      local.get 0`	`42 01`
	`      i64.const 1`	`7D`
	`      i64.sub`	`10 00`
	`      call 0`	`7E`
	`      i64.mul`	`0B`
	`  end)`	`0B 15 17`
	`; module end, size fixups`	

All integer constants are encoded using a space-efficient, variable-length LEB128 encoding.[71]

The WebAssembly text format is more canonically written in a folded format using s-expressions. For instructions and expressions, this format is purely syntactic sugar and has no behavioral differences with the linear format.[72] Through `wasm2wat`, the code above decompiles to:

```
(module
  (type $t0 (func (param i64) (result i64)))
  (func $f0 (type $t0) (param $p0 i64) (result i64)
    (if $I0 (result i64) ; $I0 is an unused label name
      (i64.eqz
        (local.get $p0)) ; the name $p0 is the same as 0 here
      (then
        (i64.const 1))
      (else
        (i64.mul
          (local.get $p0)
          (call $f0      ; the name $f0 is the same as 0 here
            (i64.sub
              (local.get $p0)
              (i64.const 1))))))))
```

Note that a module is implicitly generated by the compiler. The function is actually referenced by an entry of the type table in the binary, hence a type section and the `type` emitted by the decompiler.[73] The compiler and decompiler can be accessed online.[74]

Literature

- Haas, Andreas; Rossberg, Andreas; Schuff, Derek L.; Titzer, Ben L.; Gohman, Dan; Wagner, Luke; Zakai, Alon; Bastien, JF; Holman, Michael (June 2017). "Bringing the web up to speed with WebAssembly" (https://dl.acm.org/ft_gateway.cfm?id=3062363&type=pdf). *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. Association for Computing Machinery: 185–200. doi:10.1145/3062341.3062363 (https://doi.org/10.1145%2F3062341.3062363). ISBN 9781450349888.
- Watt, Conrad (2018). "Mechanising and Verifying the WebAssembly Specification" (https://www.cl.cam.ac.uk/~caw77/papers/mechanising-and-verifying-the-webassembly-specification.pdf) (PDF). *ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM. **7**: 53–65. doi:10.1145/3167082 (https://doi.org/10.1145%2F3167082). ISBN 9781450355865.

Notes

1. According to official documentation the Emscripten SDK may be used to create `.wasm` files which then may be executed in web browser.[9][10][11] Even though Emscripten can consume various languages when using Clang some problems may arise.[12]

References

1. "Introduction — WebAssembly 1.0" (https://webassembly.github.io/spec/core/intro/introduction.html). *webassembly.github.io*. Retrieved 18 June 2019. "WebAssembly is an open standard..."
2. "Introduction — WebAssembly 1.0" (https://webassembly.github.io/spec/core/intro/introduction.html). *webassembly.github.io*. Retrieved 18 June 2019. "WebAssembly is a ... code format"
3. "Conventions — WebAssembly 1.0" (https://webassembly.github.io/spec/core/syntax/conventions.html). *webassembly.github.io*. Retrieved 17 May 2019. "WebAssembly is a programming language that has multiple concrete representations (its binary format and the text format). Both map to a common structure."
4. "Introduction — WebAssembly 1.0" (https://webassembly.github.io/spec/core/intro/introduction.html). *webassembly.github.io*. Retrieved 18 June 2019. "... this specification is complemented by additional documents defining interfaces to specific embedding environments such as the Web. These will each define a WebAssembly application programming interface (API) suitable for a given environment."
5. "WebAssembly Specification Release 1.0 (Draft, last updated Apr 16, 2019): Introduction" (https://webassembly.github.io/spec/core/intro/introduction.html). *webassembly.org*. Retrieved 6 May 2019. "Its main goal is to enable high performance applications on the Web, but it does not make any Web-specific assumptions or provide Web-specific features, so it can be employed in other environments as well."
6. Haas, Andreas; Rossberg, Andreas; Schuff, Derek L.; Titzer, Ben L.; Holman, Michael; Gohman, Dan; Wagner, Luke; Zakai, Alon; Bastien, JF (14 June 2017). "Bringing the Web Up to Speed with WebAssembly". *SIGPLAN Notices*. **52** (6): 185–200. doi:10.1145/3140587.3062363 (https://doi.org/10.1145%2F3140587.3062363). ISSN 0362-1340 (https://www.worldcat.org/issn/0362-1340). "While the Web is the primary motivation for WebAssembly, nothing in its design depends on the Web or a JavaScript environment. It is an open standard specifically designed for embedding in multiple contexts, and we expect that stand-alone implementations will become available in the future."

7. World Wide Web Consortium. "WebAssembly Core Specification" (https://www.w3.org/TR/wasm-core-1/). *World Wide Web Consortium (W3)*. Retrieved 9 December 2019.
8. Couriol, Bruno. "WebAssembly 1.0 Becomes a W3C Recommendation and the Fourth Language to Run Natively in Browsers" (https://www.infoq.com/news/2019/12/webassembly-w3c-recommendation/). *infoq.com*. Retrieved 9 December 2019.
9. "Developer's Guide - WebAssembly" (https://webassembly.org/getting-started/developers-guide/). *webassembly.org*. Retrieved 10 June 2019.
10. "Compiling a New C/C++ Module to WebAssembly" (https://developer.mozilla.org/en-US/docs/WebAssembly/C_to_wasm). *MDN Web Docs*. Retrieved 10 June 2019.
11. "Building to WebAssembly — Emscripten 1.38.33 documentation" (https://emscripten.org/docs/compiling/WebAssembly.html). *emscripten.org*. Retrieved 10 June 2019.
12. "Emscripting a C library to Wasm | Web" (https://developers.google.com/web/updates/2018/03/emscripting-a-c-library). *Google Developers*. Retrieved 10 June 2019.
13. "stdweb - Rust" (https://docs.rs/stdweb/*/stdweb/). *docs.rs*. Retrieved 5 June 2019. "The goal of this crate is to provide Rust bindings to the Web APIs and to allow a high degree of interoperability between Rust and JavaScript."
14. "web_sys - Rust" (https://docs.rs/web-sys/*/web_sys/). *docs.rs*. Retrieved 5 June 2019. "Raw API bindings for Web APIs. This is a procedurally generated crate from browser WebIDL which provides a binding to all APIs that browser provide on the web."
15. "js_sys - Rust" (https://docs.rs/js-sys/*/js_sys/). *docs.rs*. Retrieved 5 June 2019. "Bindings to JavaScript's standard, built-in objects, including their methods and properties."
16. Bright, Peter (18 June 2015). "The Web is getting its bytecode: WebAssembly" (https://arstechnica.com/information-technology/2015/06/the-web-is-getting-its-bytecode-webassembly/). *Ars Technica*. Condé Nast.
17. "New Bytecode Alliance Brings the Security, Ubiquity, and Interoperability of the Web to the World of Pervasive Computing" (https://blog.mozilla.org/press/2019/11/new-bytecode-alliance-brings-the-security-ubiquity-and-interoperability-of-the-web-to-the-world-of-pervasive-computing/). *Mozilla*. 12 November 2019. Retrieved 27 May 2019.
18. "Launch bug" (https://github.com/WebAssembly/design/issues/150). *GitHub / WebAssembly / design*. 11 June 2015.
19. Wagner, Luke (14 March 2016). "A WebAssembly Milestone: Experimental Support in Multiple Browsers" (https://hacks.mozilla.org/2016/03/a-webassembly-milestone/). *Mozilla Hacks*.
20. Thompson, Seth (15 March 2016). "Experimental support for WebAssembly in V8" (https://v8project.blogspot.com/2016/03/experimental-support-for-webassembly.html). *V8 Blog*.
21. Zhu, Limin (15 March 2016). "Previewing WebAssembly experiments in Microsoft Edge" (https://blogs.windows.com/msedgedev/2016/03/15/previewing-webassembly-experiments/). *Microsoft Edge dev blog*.
22. Lardinois, Frederic (17 June 2015). "Google, Microsoft, Mozilla And Others Team Up To Launch WebAssembly, A New Binary Format For The Web" (https://techcrunch.com/2015/06/17/google-microsoft-mozilla-and-others-team-up-to-launch-webassembly-a-new-binary-format-for-the-web/). *TechCrunch*. Retrieved 24 December 2017.
23. Avram, Abel (31 May 2017). "Google Is to Remove Support for PNaCl" (https://www.infoq.com/news/2017/05/pnacl-webassembly-google). *InfoQ*. Retrieved 22 December 2017.
24. "WebAssembly: a binary format for the web" (https://www.2ality.com/2015/06/web-assembly.html). *②ality – JavaScript and more*. 18 June 2015.
25. "Staring at the Sun: Dalvik vs. ASM.js vs. Native" (https://blog.mozilla.org/javascript/2013/08/01/staring-at-the-sun-dalvik-vs-spidermonkey/). *blog.mozilla.org*. Retrieved 7 December 2019. "Even discarding the one score where asm.js did better, it executes at around 70% of the speed of native C++ code."
26. Krill, Paul (6 March 2017). "WebAssembly is now ready for browsers to use" (https://www.infoworld.com/article/3176681/). *InfoWorld*. Retrieved 23 December 2017.

27. "WebAssembly First Public Working Drafts" (https://www.w3.org/blog/news/archives/6838). W3C. 15 February 2018. Retrieved 20 April 2018.
28. "WebAssembly Core Specification" (https://www.w3.org/TR/2018/WD-wasm-core-1-20180215/). W3C. 15 February 2018. Retrieved 20 April 2018.
29. "WebAssembly JavaScript Interface" (https://www.w3.org/TR/2018/WD-wasm-js-api-1-20180215/). W3C. 15 February 2018. Retrieved 20 April 2018.
30. "WebAssembly Web API" (https://www.w3.org/TR/2018/WD-wasm-web-api-1-20180215/). W3C. 15 February 2018. Retrieved 20 April 2018.
31. "WebAssembly support now shipping in all major browsers" (https://blog.mozilla.org/blog/2017/11/13/webassembly-in-browsers/). *The Mozilla Blog*. Retrieved 21 November 2017.
32. "Introducing new JavaScript optimizations, WebAssembly, SharedArrayBuffer, and Atomics in EdgeHTML 16" (https://blogs.windows.com/msedgedev/2017/10/31/optimizations-webassembly-sharedarraybuffer-atomics-edgehtml-16/). *Microsoft Edge Dev Blog*. 31 October 2017. Retrieved 21 November 2017.
33. "WebAssembly" (https://caniuse.com/wasm). *Can I use*. Retrieved 25 May 2020.
34. Bright, Peter (18 June 2015). "The Web is getting its bytecode: WebAssembly" (https://arstechnica.com/information-technology/2015/06/the-web-is-getting-its-bytecode-webassembly/). *Ars Technica*. Retrieved 23 December 2017.
35. Ball, Kevin (26 June 2018). "How WebAssembly is Accelerating the Future of Web Development" (https://web.archive.org/web/20190212141715/https://zendev.com/2018/06/26/webassembly-accelerating-future-web-development.html). Archived from the original (https://zendev.com/2018/06/26/webassembly-accelerating-future-web-development.html) on 12 February 2019. Retrieved 22 October 2018.
36. "Awesome WebAssembly Languages" (https://web.archive.org/web/20190212133939/https://github.com/appcypher/awesome-wasm-langs). 26 June 2018. Archived from the original (https://github.com/appcypher/awesome-wasm-langs) on 12 February 2019. Retrieved 12 February 2019.
37. Zakai, Alon [@kripken] (21 October 2019). "Emscripten has switched to the upstream LLVM wasm backend by default! / Details:https://groups.google.com/forum/#!topic/emscripten-discuss/NpxVAOirSl4 …" (https://twitter.com/kripken/status/1186407352880074752) (Tweet). Retrieved 22 October 2019 – via Twitter.
38. "LLVM 8.0.0 Release Notes — LLVM 8 documentation" (https://releases.llvm.org/8.0.0/docs/ReleaseNotes.html#changes-to-the-webassembly-target). *releases.llvm.org*. Retrieved 22 October 2019.
39. "WebAssembly High-Level Goals" (https://github.com/WebAssembly/design/blob/master/HighLevelGoals.md). *GitHub / WebAssembly / design*. 11 December 2015.
40. Krill, Paul (29 November 2017). "Direct WebAssembly compilation comes to Rust language" (https://www.infoworld.com/article/3239129/). *InfoWorld*. Retrieved 24 December 2017.
41. "Frequently asked questions (FAQ) about Blazor" (https://blazor.net/docs/introduction/faq.html). *blazor.net*. Retrieved 18 June 2018.
42. Krill, Paul (26 October 2017). "What's next for WebAssembly: GC, threads, debugging" (https://www.techworld.com.au/article/629123/). *TechWorld*. Retrieved 24 December 2017.
43. " Garbage collection · Issue #16 · WebAssembly/proposals" (https://github.com/WebAssembly/proposals/issues/16). *GitHub*. Retrieved 25 July 2019.
44. "Bolero: F# in WebAssembly" (https://fsbolero.io/). *fsbolero.io*. Retrieved 25 July 2019.
45. "Julia in the Browser" (https://nextjournal.com/sdanisch/wasm-julia). *nextjournal.com*. Retrieved 9 April 2019.
46. "WebAssembly platform by tshort · Pull Request #2 · JuliaPackaging/Yggdrasil" (https://github.com/JuliaPackaging/Yggdrasil/pull/2). *GitHub*. Retrieved 9 April 2019.
47. Fischer, Keno (22 July 2019), *GitHub - Keno/julia-wasm: Running julia on wasm.* (https://github.com/Keno/julia-wasm), retrieved 25 July 2019

48. "MRuby in Your Browser" (https://ruby.dj/). *ruby.dj*. Retrieved 25 July 2019.
49. Neumann, Robert; Toro, Abel (19 April 2018). "In-browser mining: Coinhive and WebAssembly" (https://www.forcepoint.com/blog/x-labs/browser-mining-coinhive-and-webassembly). Forcepoint. Retrieved 8 June 2019.
50. Cimpanu, Catalin (24 June 2018). "Changes in WebAssembly Could Render Meltdown and Spectre Browser Patches Useless" (https://www.bleepingcomputer.com/news/security/changes-in-webassembly-could-render-meltdown-and-spectre-browser-patches-useless/). Bleeping Computer. Retrieved 8 June 2019.
51. Sanders, James (25 June 2018). "How opaque WebAssembly code could increase the risk of Spectre attacks online" (https://www.techrepublic.com/article/how-opaque-webassembly-code-could-increase-the-risk-of-spectre-attacks-online/). *Tech Republic*. Retrieved 9 June 2019.
52. R, Bhagyashree (30 October 2018). "Google Chrome 70 now supports WebAssembly threads to build multi-threaded web applications" (https://hub.packtpub.com/google-chrome-70-now-supports-webassembly-threads-to-build-multi-threaded-web-applications/). *Packt Pub*. Retrieved 9 June 2019.
53. Lonkar, Aishwarya; Chandrayan, Siddhesh (October 2018). "The dark side of WebAssembly" (https://www.virusbulletin.com/virusbulletin/2018/10/dark-side-webassembly/). *Virus Bulletin*. Retrieved 8 June 2019.
54. Segura, Jérôme (29 November 2017). "Persistent drive-by cryptomining coming to a browser near you" (https://blog.malwarebytes.com/cybercrime/2017/11/persistent-drive-by-cryptomining-coming-to-a-browser-near-you/). Malwarebytes. Retrieved 8 June 2019.
55. "Recent Study Estimates That 50% of Websites Using WebAssembly Apply It for Malicious Purposes" (https://www.infoq.com/news/2019/10/WebAssembly-wasm-malicious-usage/). *InfoQ*. Retrieved 3 November 2019.
56. Musch, Marius; Wressnegger, Christian; Johns, Martin; Rieck, Konrad (June 2019). "New Kid on the Web: A Study on the Prevalence of WebAssembly in the Wild" (https://www.tu-braunschweig.de/Medien-DB/ias/pubs/2019-dimva.pdf) (PDF). *Detection of Intrusions and Malware, and Vulnerability Assessment*. Lecture Notes in Computer Science. **11543**. Detection of Intrusions and Malware, and Vulnerability Assessment. pp. 23–42. doi:10.1007/978-3-030-22038-9_2 (https://doi.org/10.1007%2F978-3-030-22038-9_2). ISBN 978-3-030-22037-2. Retrieved 4 November 2019. Slides (PDF) (https://www.dimva2019.org/wp-content/uploads/sites/31/2019/06/DIMVA19-slides-2-R.pdf)
57. "Symbolically Executing WebAssembly in Manticore" (https://blog.trailofbits.com/2020/01/31/symbolically-executing-webassembly-in-manticore/). 31 January 2020. Retrieved 10 February 2020.
58. "Non-Web Embeddings" (https://webassembly.org/docs/non-web/). *WebAssembly*. Retrieved 15 May 2019.
59. "Non-Web Embeddings" (https://github.com/WebAssembly/design/blob/master/NonWeb.md). *GitHub / WebAssembly*. Retrieved 15 May 2019.
60. "WebAssembly System Interface Repo" (https://github.com/WebAssembly/WASI). *GitHub / WebAssembly*. 10 February 2020.
61. "Additional background on Capabilities" (https://github.com/bytecodealliance/wasmtime/blob/master/docs/WASI-capabilities.md). *GitHub*. bytecodealliance.
62. "Standardizing WASI: A system interface to run WebAssembly outside the web – Mozilla Hacks - the Web developer blog" (https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/). *Mozilla Hacks – the Web developer blog*.
63. "reference-sysroot Repo" (https://github.com/WebAssembly/reference-sysroot). *GitHub / WebAssembly*. 12 January 2020.
64. "wasm-c-api Repo" (https://github.com/WebAssembly/wasm-c-api). *GitHub / WebAssembly*. 3 February 2020.

65. "Design Rationale" (https://github.com/WebAssembly/design/blob/master/Rationale.md#why-a-stack-machine). *GitHub / WebAssembly / design*. 1 October 2016.
66. "Conventions — WebAssembly 1.0" (https://webassembly.github.io/spec/core/syntax/conventions.html). *webassembly.github.io*. Retrieved 12 November 2019.
67. "Introduction — WebAssembly 1.0" (https://webassembly.github.io/spec/core/intro/introduction.html?highlight=isa#scope). *webassembly.github.io*. Retrieved 17 May 2019.
68. "Instructions — WebAssembly 1.0" (https://webassembly.github.io/spec/core/syntax/instructions.html). *webassembly.github.io*. Retrieved 12 November 2019.
69. "Roadmap" (https://webassembly.org/roadmap/). *WebAssembly*. March 2017.
70. jfbastien; rossberg-chromium; kripken; titzer; s3ththompson; sunfishcode; lukewagner; flagxor; enricobacis; c3d; binji; andrewosh (9 March 2017). "Text Format" (https://github.com/WebAssembly/design/blob/master/TextFormat.md). *WebAssembly/design*. GitHub.
71. WebAssembly Community Group (January 2020). "WebAssembly Specification Release 1.0" (https://webassembly.github.io/spec/core/binary/values.html#integers). Retrieved 13 January 2020.
72. "Folded instructions" (https://webassembly.github.io/spec/core/text/instructions.html#folded-instructions). *GitHub*. / WebAssembly / spec
73. "Modules (Binary)" (https://webassembly.github.io/spec/core/syntax/modules.html). *WebAssembly 1.0*.
74. "WebAssembly Binary Toolkit (wabt) demos" (https://webassembly.github.io/wabt/demo/). *webassembly.github.io*.

Demo

- Widgets demo: [1] (https://yutakaaoki.github.io/demo1/index.html) with NWSTK
- 3D mountain geometry synthesis demo: [2] (https://yutakaaoki.github.io/demo_Mountain/index.html) with NWSTK
- Demo for loading and drawing a jpg file: [3] (https://yutakaaoki.github.io/demo2/index.html) with NWSTK

External links

- Official website (https://webassembly.org/)
- W3C Community Group (https://www.w3.org/community/webassembly/)
- WebAssembly Design (https://github.com/WebAssembly/design)
- "WebAssembly" (https://developer.mozilla.org/en-US/docs/WebAssembly), *MDN Web Docs* – with info on browser compatibility and specifications (WebAssembly JavaScript API)

Foundation, Inc., a non-profit organization.