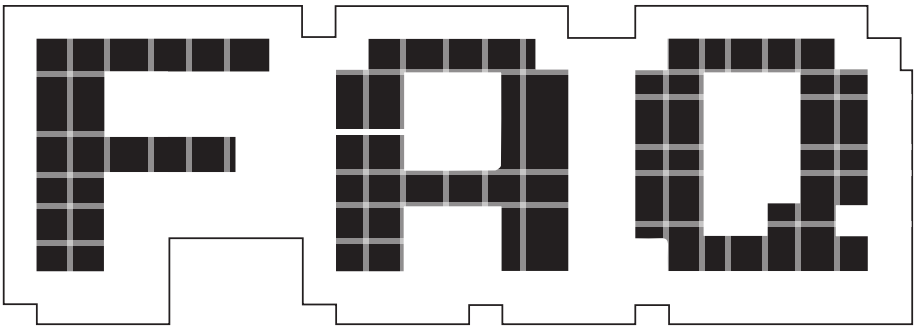

ASM.JS

Bringing near-native performance to cross-platform web apps.

BEN EVERARD

Q ASM usually means assembly, and .js usually means a JavaScript library, but what on earth are those two things doing together?

A The idea behind asm.js is to remove everything from JavaScript that doesn't run quickly. The result is a very strict subset of JavaScript that isn't as nice to program in, but does run much faster.

Although asm.js is an interpreted language that you could program in, programmers will usually write code in another language, and then compile that language to asm.js. In other words, asm.js is designed to be a little like assembly language, but it's actually JavaScript.

Q So it's just another JavaScript engine?

A No. In essence, asm.js isn't anything other than a specification of a subset of JavaScript.

> **"The idea of asm.js is to minimise performance concerns as much as possible."**

Any JavaScript code that only uses this subset can be said to use asm.js. However, since it's valid JavaScript, it will run on any JavaScript engine. As there are JavaScript engines for almost every computing platform built in the past decade, compiling code to asm.js means it should be very cross-plaform.

Q Wait a minute: compiling code to an intermediate language so it can run on web browsers… this sounds a lot like Java! Do we really need another option for this?

A There is a certain similarity in the concepts behind Java and asm.js. However, they're solutions designed for different ages. Java applets are placed on a page and given a certain area that they are allowed to interact with. In other words, they were a single item on a larger page. This means that, while they have some uses, they have never really been suitable for full-on web apps.

JavaScript (and by extension asm.js) can interact with the entire web page. It can add, remove and manipulate items in the HTML in an almost endless series of ways. In other words, JavaScript can be used to control the entire web page. This makes it a much better option for modern web apps. What's more, almost all modern web browsers support JavaScript out of the box without the need for plugins. This means that you don't need to ask users to download anything in order to access your site.

Q Why bother using a subset of JavaScript, when the full language is already supported on multiple browsers on most modern operating systems?

A The advantage of limiting the available options is speed. Using *Firefox* (which is the browser that handles asm.js best), the JavaScript engine is able to detect when a particular script is written in asm.js, and will optimise itself accordingly. This gives it the advantage of running everywhere (because it is a subset of JavaScript), yet being able to run very quickly when the JavaScript engine supports it.

Q So it already works everywhere? Does that mean I don't need to change my browser?

A It's not essential, but like we said before, *Firefox* optimises itself when it detects asm.js code. This means that it will run much faster on recent versions of *Firefox* than it will on other browsers. Obviously, speed isn't always essential, but when it is, you're

better off using *Firefox* for asm.js web apps. *Chrome*, after a slow start, is catching up. Other browsers are likely to perform worse at the moment, but may well see improvements in time.

Q You mentioned earlier that programmers write in other languages, and then compile to JavaScript. What languages can you program asm.js in?

A So far, most of the work has focused on C and C++. The support for both of these is provided through the *Emscripten* source-to-source compiler. Since a large proportion of computer games are written in these languages, ams.js has been used to port games to the web (using WebGL for graphics). Perhaps the most famous asm.js project is the port of the *Unity* games engine (for example, *Dead Trigger 2* – **http://beta.unity3d.com/jonas/DT2** and *AngryBots* – **http://beta.unity3d.com/jonas/AngryBots**).

However, support for other languages is coming. Python has some support (via pypy.js), and the Lua VM can be built through *Emscripten*, but neither of these are really at the level of the C and C++ versions yet.

Q Why not just skip this step and write in JavaScript?

A There are a few reasons! There's obviously a lot of legacy code that exists already in C and C++, so why bother re-writing it in JavaScript if you can just compile it? You might want a single codebase that can compile to both native and browser. Also, compiled asm.js code tends to be quite a bit faster than hand-writen JavaScript because it takes advantage of a whole host of optimisations.

Q It looks to me a little like most of the advantages of asm.js happen when you take something that is normally a native app and convert it into a web app. Isn't this a bad idea? I mean, wouldn't it be better just to compile the C or C++ to native code?

A Whether or not it's a good idea depends on many things, but basically it's always a trade-off. Putting software in web apps can make them easier to access across a range of devices, but on the other hand, there are privacy and security concerns, and performance can be a problem. The idea of asm.js is to minimise the performance concerns as much as possible. In fact, benchmarks show that code compiled to asm.js can run at about twice the speed of the same code compiled natively. This might sound like quite a big slowdown, but it doesn't mean that programs will run at half the speed, because only a small proportion of most software is actually waiting for a bit of code to run. Most of the time the computer's waiting for user input, or for some data to be retrieved from the disk, or (in the case of games) a 3D scene to render on the graphics card. This means that plenty of software will appear to run at the same speed when using asm.js as when compiled to native code. This doesn't change the trade-off between access on multiple devices and security, which will be highly dependent on the application and who's hosting it.

We should also point out here that although JavaScript is usually used for web apps, you don't have to use it this way. There's nothing to stop you using asm.js to create software that doesn't rely on the network, and just uses the JavaScript engine to provide portability. If asm.js takes off, we're likely to see more and more software doing this. In fact, it's already possible to compile some *Qt* software to asm.js. There are some examples at **http://vps2.etotheipiplusone.com:30176/redmine/projects/emscripten-qt/wiki/Demos**.

If something as computationally intense as an FPS game can run in asm.js, then most other software should have no problem.

Q This all sounds wonderful. How can I compile my C and C++ programs to asm.js?

Software can be compiled to asm.js using *Emscripten*. Use this in exactly the same way you would any other compiler. Asm.js is used when you set the optimisation flag to **-01** or higher. This can output pure JavaScript or an HTML file that includes the JavaScript. See the tutorial at **http://kripken.github.io/emscripten-site/docs/getting_started/Tutorial.html** for a useful look at how to get started.

Q What about stuff that JavaScript in the browser just can't do, like access the filesystem, and link to libraries.

A There's no way that asm.js can access the filesystem of a machine when running on a website – JavaScript is deliberately kept separate from the machine it's running on for security reasons. However, asm.js programs can access a virtual file system. This enables the developer to use the same C and C++ code that accesses files, but at the same time, still protect the host machine from any malicious asm.js code.

Libraries are another matter. By default, asm.js includes libc, libc++ and SDL. If you want to work with other libraries, you could try compiling those libraries to asm.js, or re-implementing the features you need. There's some more details on this on the *Emscripten* FAQ: **http://kripken.github.io/emscripten-site/docs/getting_started/FAQ.html**