asm.js: Semantic additions for shared memory and atomics

Stable draft, 21 November 2014
Updated, 19 January 2015

Note 2015-08-28: This specification is **OBSOLETE**. It has **MOVED TO GITHUB**. All open comment threads have been replicated as individual Issues on the moved spec.

At a glance:

- New constructors "Shared{Int,Uint}{8,16,32}Array" can be mapped onto the heap if the heap is a SharedArrayBuffer.
- It is a static error to reference, even for the purposes of storing it in a local binding, a shared-view constructor (eg, SharedInt32Array) and an unshared-view constructor (eg, Float64Array) in the same compilation unit.
- It is a link error to map a shared view onto an unshared buffer, or an unshared view onto a shared buffer. [2015/01/19]
- Loading from and storing to the heap have unchanged semantics if the heap is shared memory.
- There is a new known stdlib object "Atomics"
- There are new known methods "load", "store", "fence", "compareExchange", "add", "sub", "and", "or", and "xor" on the Atomics object.
- For all those operations except "fence":
    - The first argument must name a shared integer-typed array
    - The second argument must be an intish expression:
        - It signifies the element index within the array
        - It must satisfy the same constraints as a heap access expression: if the array is not a byte array then the expression must have the form c, where c is a constant, or the form e>>k, where k is a constant that is the log-base-2 of the element size.[1]
        - If the element index is not in the range of the array then
            - A memory barrier appropriate to the operation is performed
            - No read or write operation is performed
            - For load, compareExchange, and the binary operators the result value is zero
- Meaning of Atomics.load()
    - The result type is signed[2]
- Meaning of Atomics.store()
    - The third argument must be intish
    - The result value is the third argument value, coerced to signed

[1] Treating atomic accesses as "syntax" rather than as "calls" fits in with how they will be used and allows for simplifictions in code generation, and is not (yet) seen as a particular hardship. There will be some spec complexity regardless.
[2] In principle it should be intish but that is not an allowed result type for a call.

    - The result type is signed[3]
- Meaning of Atomics.fence()
    - The result type is void
- Meaning of Atomics.compareExchange()
    - The third argument must be intish
    - The fourth argument must be intish
    - The result type is signed
- Meaning of Atomics.add(), Atomics.sub(), Atomics.and(), Atomics.or(), Atomics.xor()
    - The third argument must be intish
    - The result type is signed
- Meaning of Atomics.isLockFree
    - The argument must be an integer constant
    - The result type is int (a boolean value)

The following Atomics names are *not* available as intrinsics in asm.js:

Atomics.futexWait()
Atomics.futexWake()
Atomics.futexWakeOrRequeue()
Atomics.OK
Atomics.NOTEQUAL
Atomics.TIMEDOUT

The futex methods can be accessed through the FFI (with the heap passed implicitly) and the result values can be expanded into constant values, as they have known values.

[3] In principle it should perhaps be the concrete type of the third argument, which would be the case for an assignment to a typed array element, but this is a call and it needs a proper call return type