Simplifying Bulk Data Instructions?

Andreas Rossberg
Dfinity

Context

Bulk instructions test some extra conditions:

- all: is the memory/table access out of bounds?

- copy: do the memory/table ranges overlap?

These could potentially be simplified

Benefits: smaller code, simpler spec

Both simplifications are independently useful

Overlap

Copy needs to work for overlapping ranges

Current condition for reverse copy:

$src < dst \ \wedge \ src + len > dst$

Could be simplified to just:

$src < dst$

Difference only observable by OOB or race, both of which is undefined behaviour in C

Precedence

glibc, musl, newlib: stricter condition

llvm, freebsd libc: simpler condition

Benefits of simpler

Slightly less code to generate and execute

Simpler spec (can unroll incrementally, no “administrative” instruction needed)

Benefits of stricter

Reverse slower on some hardware??

Out of Bounds

Currently, checking for out of bounds – even for length zero

(**memory.copy** (-1) (-1) (0)) ↪ **trap**

Inherited from earlier segment check

But those were static!

At runtime, extra code and dynamic cost for each execution; also, potential concurrency

Semantics

Three cases for (**memory.fill** *src val len*)

1.) *len* > 0: **store** *val* to [*src*, *src*+*len*)

2.) *len* = 0 ∧ *src*+*len* ≤ *limit*: **nop**

3.) *len* = 0 ∧ *src*+*len* > *limit*: **trap**

Appear in both generated code and spec

Proposal: eliminate (2)+(3), always do (1)

Semantics

Three cases for (**memory.fill** *src val len*)

store *val* to [*src*, *src*+*len*)

Appear in both generated code and spec

Proposal: eliminate (2)+(3), always do (1)

Benefits

Simpler, removes special cases (in fact, cases)

Less code to generate and execute

More modular spec,
especially in presence of shared memories

…can express all bulk ups in terms of simple ops
…agnostic to shared vs non-shared

Less surprising?

Upshot

Simple equivalences (for shared & non-shared):

(memory.fill D V N) =
  (i32.store8 D V) … (i32.store8 (D+N-1) V)

(memory.copy D S N) =
  (i32.store8 D (i32.load8 S)) … (i32.store8 (D+N-1) (i32.load8 (S+N-1)))

…etc…

Useful for code transformations

Disadvantages

Allows some (harmless) additional cases