WASM Trace Proposal

July 20, 2021

Jacob Abraham and Rich Winterton

Proposal

- “instTrace” custom section
- Contains a list of trace instructions
  - Each instruction is encoded as a byte offset into the code section followed by an immediate ID
  - Immediate ID allows traces to be identified easily
  - No need to specify instruction in WASM
  - Requires the target implementor to know which native code sequence
- Intrinsic
  - __builtin_wasm_trace_instruction(ID)
  - Prototype in clang
- Runtime
  - Decode and inserted in instruction stream
  - Prototype in V8

C Source Example

- Should add a custom section
  - 2 traces, one before the i32.add and one after

```
int add(int a, int b) {
  __builtin_wasm_trace_instruction(17);
  int c = a+b;
  __builtin_wasm_trace_instruction(18);
  return c;
}
```

ASM Example

clang --target=wasm32 -c test.c -S -O3 -o test.s

What the prototype generates

```
add:
  .functype  add (i32, i32) -> (i32)
.Ltrace0:
  local.get  1
  local.get  0
  i32.add
  local.set 1
.Ltrace1:
  local.get 1
  end_function
```

Custom section defining the labels

```
.section  .custom_section.instTrace,"",@
  .int8   2
  .int32  .Ltrace0
  .int8   17
  .int32  .Ltrace1
  .int8   18
```

WASM Example

clang --target=wasm32 -c -emit-llvm test.c –O3 -o test.bc

llc -march=wasm32 -filetype=obj test.bc

wasm-ld --no-entry --export=add test.o -o test.wasm

Ideally what we want

```
Contents of section Custom:
0969 6e73 7454 7261 6365     len, instTrace
0203 110a 12                 num traces,
                             offset, immediate,
                             offset, immediate
```

Ideally, we want uvar32 for offsets, limitation in clang

What the prototype generates

```
Contents of section Custom:
0969 6e73 7454 7261 6365     len, instTrace
0203 0000 0011 0a00 0000 12  num traces,
                             offset, immediate,
                             offset, immediate
```

*discussed in issues

Runtime Code Gen

Experiment running in d8, using only Liftoff

TurboFan work in progress*

d8 --experimental-wasm-instruction-tracing --print-code --liftoff-only test.js

With runtime flag “--wasm-trace-native=<type>”, we can switch what native code is run for different simulators.

Minimal overhead, on par with optimized debug builds

*discussed in issues

```
00000350B49413C0     0  55              push rbp
00000350B49413C1     1  4889e5          REX.W movq rbp,rsp
00000350B49413C4     4  6a08            push 0x8
00000350B49413C6     6  4881ec08000000 REX.W subq rsp,0x8
00000350B49413CD     d  488975f0        REX.W movq [rbp-0x10],rsi
00000350B49413D1    11  488b5e23        REX.W movq rbx,[rsi+0x23]
00000350B49413D5    15  483b23          REX.W cmpq rsp,[rbx]
00000350B49413D8    18  0f864a000000    jna 00000350B4941428  <+0x68>
00000350B49413DE    1e  53              push rbx
00000350B49413DF    1f  bb11000000      movl rbx,0000000000000011
00000350B49413E4    24  6467909090      sscmark
00000350B49413E9    29  5b              pop rbx
00000350B49413EA    2a  8d1c02          leal rbx,[rdx+rax*1]
00000350B49413ED    2d  53              push rbx
00000350B49413EE    2e  bb12000000      movl rbx,0000000000000012
00000350B49413F3    33  6467909090      sscmark
00000350B49413F8    38  5b              pop rbx
00000350B49413F9    39  8bc3            movl rax,rbx
00000350B49413FB    3b  48837df808      REX.W cmpq [rbp-0x8],0x8
00000350B4941400    40  7421            jz 00000350B4941423  <+0x63>
00000350B4941402    42  b924000000      movl rcx,0000000000000024
00000350B4941407    47  4989e2          REX.W movq r10,rsp
00000350B494140A    4a  4883ec28        REX.W subq rsp,0x28
00000350B494140E    4e  4883e4f0        REX.W andq rsp,0xf0
00000350B4941412    52  4c89542420      REX.W movq [rsp+0x20],r10
00000350B4941417    57  48b84043eb25f67f0000 REX.W movq rax,00007FF625EB4340
00000350B4941421    61  ffd0            call rax
00000350B4941423    63  488be5          REX.W movq rsp,rbp
00000350B4941426    66  5d              pop rbp
00000350B4941427    67  c3              retl
00000350B4941428    68  50              push rax
00000350B4941429    69  51              push rcx
00000350B494142A    6a  52              push rdx
00000350B494142B    6b  56              push rsi
00000350B494142C    6c  e84ffeffff      call 00000350B4941280
00000350B4941431    71  5e              pop rsi
00000350B4941432    72  5a              pop rdx
00000350B4941433    73  59              pop rcx
00000350B4941434    74  58              pop rax
00000350B4941435    75  eba7            jmp 00000350B49413DE  <+0x1e>
00000350B4941437    77  90              nop
```

DevTools Enabling

- SDE (Software Development Emulator)
- Browser enabling
  - Debug breakpoints
  - Start/Stop performance trace

sde-mix-out.txt	14,016 KB
sde-mix-out-partial.txt	103 KB

Debug?	Mark	Location
☐	17	Line 3 in ee9051d6
☐	18	Line 3 in 18b8d8c6
☐	18	Line 4 in 18b8d8c6
☑	2	Line 3 in b17e684a
☐	2	Line 8 in b17e684a

b17e684a ×

```
0x000 (module
0x01d   (func $any (;0;) (export "any")
0x020     ittnop 2
0x023     i32.const 10
0x025     i32.const 7
0x027     i32.add
0x028     drop
0x029     ittnop 2
0x02c   )
0x02d )
```

Current Issues With Prototype

- V8
  - In Liftoff, trace instruction is emitted where the offset is.
  - In TurboFan, inside of a function if there are more than one trace instruction, all but the first get optimized away.
- Emscripten
  - The compilation works fine (just clang), but the optimization step with binaryen sometimes causes problems. When the WASM gets reordered, the instTrace offsets don’t move with it.

Summary

- Instruction Tracing
  - Custom section defines where trace instructions should be emitted
  - Decision made at the engine implementation
  - Compiler intrinsic to generate custom section
- https://github.com/WebAssembly/instrument-tracing

- Vote for Phase 2

- Next Steps in Phase 2
  - Support from engine teams to integrate prototype
  - Support from tools teams
    - Clang uvar32 support
  - Support for development on DevTools
  - Feedback from early adopters, including additional testing