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
  - Don’t 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

- 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

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

Ideally what we want

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

What the prototype generates

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

*discussed in issues

Custom section defining the labels

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

WASM

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
0205 0000 0011 0a00 0000 12  num traces,
                             offset, immediate,
                             offset, immediate
```

What the prototype generates

```
Contents of section Custom:
0969 6e73 7454 7261 6365     len, instTrace
0205 0000 0011 0500 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

*discussed in issues

```
000001043AE813E0     0  55             push rbp
000001043AE813E1     1  4889e5         REX.W movq rbp,rsp
000001043AE813E4     4  6a08           push 0x8
000001043AE813E6     6  4881ec08000000 REX.W subq rsp,0x8
000001043AE813ED     d  488975f0       REX.W movq [rbp-0x10],rsi
000001043AE813F1    11  488b4e23       REX.W movq rcx,[rsi+0x23]
000001043AE813F5    15  483b21         REX.W cmpq rsp,[rcx]
000001043AE813F8    18  0f864a000000   jna 000001043AE81448  <+0x68>
000001043AE813FE    1e  53             push rbx
000001043AE813FF    1f  bb11000000     movl rbx,0000000000000011
000001043AE81404    24  64             'Unimplemented Instruction'
000001043AE81405    25  67             'Unimplemented Instruction'
000001043AE81406    26  90             nop
000001043AE81407    27  90             nop
000001043AE81408    28  90             nop
000001043AE81409    29  5b             pop rbx
000001043AE8140A    2a  53             push rbx
000001043AE8140B    2b  bb12000000     movl rbx,0000000000000012
000001043AE81410    30  64             'Unimplemented Instruction'
000001043AE81411    31  67             'Unimplemented Instruction'
000001043AE81412    32  90             nop
000001043AE81413    33  90             nop
000001043AE81414    34  90             nop
000001043AE81415    35  5b             pop rbx
000001043AE81416    36  8d0c02         leal rcx,[rdx+rax*1]
000001043AE81419    39  8bc1           movl rax,rcx
000001043AE8141B    3b  48837df808     REX.W cmpq [rbp-0x8],0x8
000001043AE81420    40  7421           jz 000001043AE81443  <+0x63>
000001043AE81422    42  b924000000     movl rcx,0000000000000024
000001043AE81427    47  4989e2         REX.W movq r10,rsp
000001043AE8142A    4a  4883ec28       REX.W subq rsp,0x28
000001043AE8142E    4e  4883e4f0       REX.W andq rsp,0xf0
000001043AE81432    52  4c89542420     REX.W movq [rsp+0x20],r10
000001043AE81437    57  48b840431335f77f0000 REX.W movq rax,00007FF735134340
000001043AE81441    61  ffd0           call rax
000001043AE81443    63  488be5         REX.W movq rsp,rbp
000001043AE81446    66  5d             pop rbp
000001043AE81447    67  c3             retl
000001043AE81448    68  50             push rax
000001043AE81449    69  52             push rdx
000001043AE8144A    6a  56             push rsi
000001043AE8144B    6b  e830feffff     call 000001043AE81280
000001043AE81450    70  5e             pop rsi
000001043AE81451    71  5a             pop rdx
000001043AE81452    72  58             pop rax
000001043AE81453    73  eba9           jmp 000001043AE813FE  <+0x1e>
000001043AE81455    75  0f1f00         nop
```

Current Issues

- Clang
  - There are certain optimizations after the translation to LLVM IR that cause the trace instructions to be moved to the top of the function. The LLVM IR generates correctly, but as you can see in the ASM this is not the case.
- 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

- Vote for Phase 2