如何调试V8 学习报告

来源：v8-riscv.v8.wiki

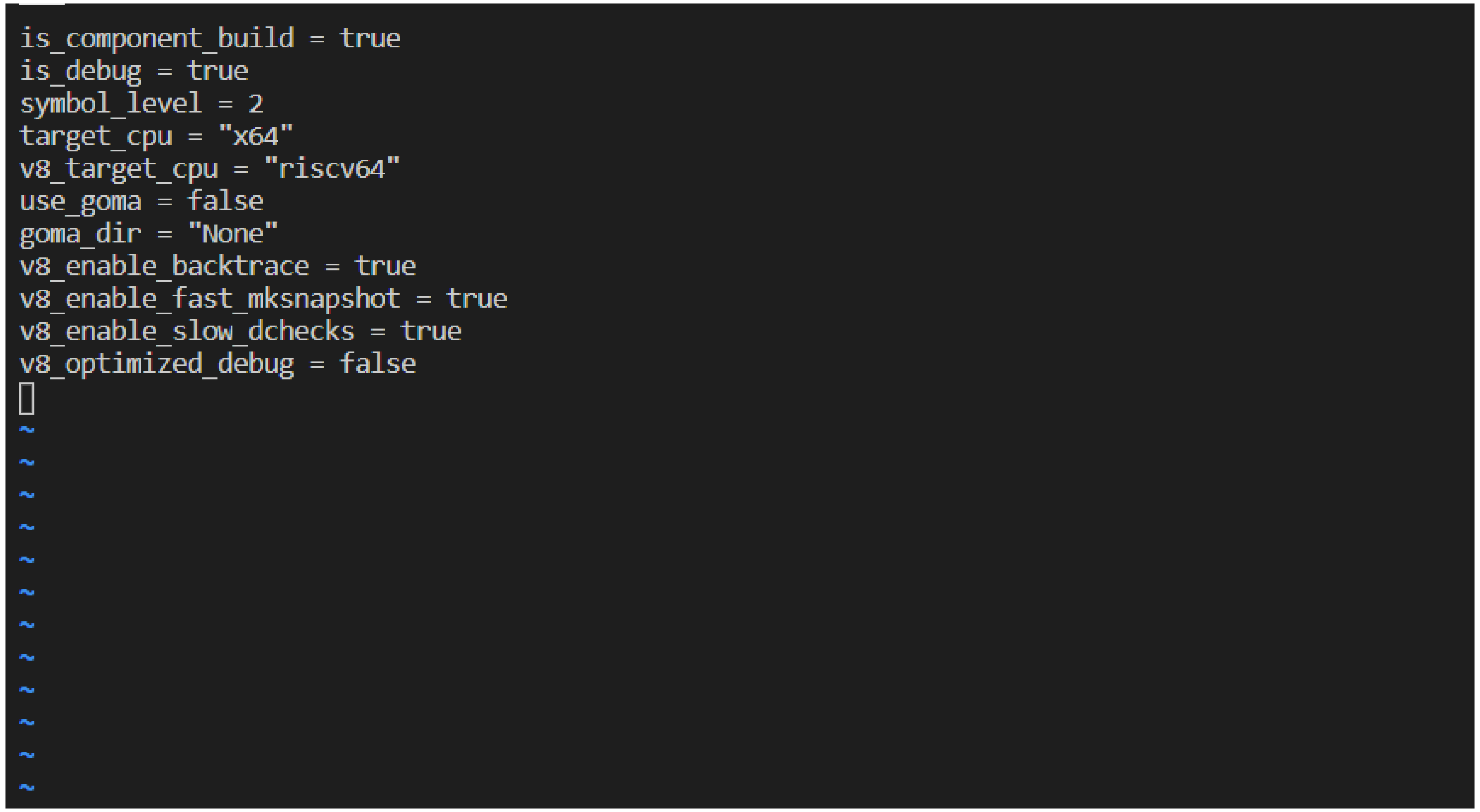
报告人：梁斌

整体策略

1.简化测试用例
2.确定基本事实
3.寻找线索
4.将执行跟踪映射回V8源代码

参数配置

Args.gm文件：



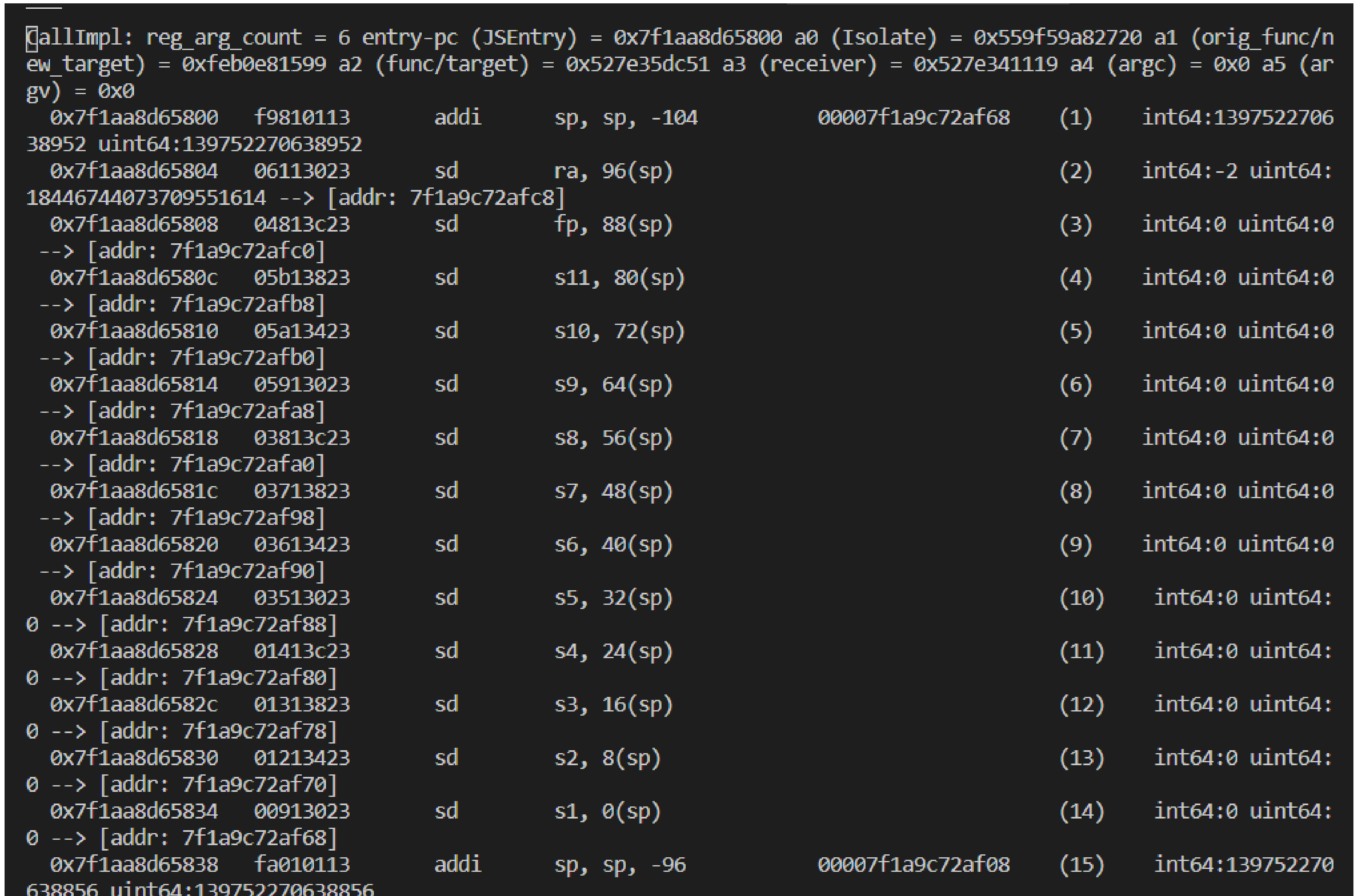
```
is_component_build = true
is_debug = true
symbol_level = 2
target_cpu = "x64"
v8_target_cpu = "riscv64"
use_goma = false
goma_dir = "None"
v8_enable_backtrace = true
v8_enable_fast_mksnapshot = true
v8_enable_slow_dchecks = true
v8_optimized_debug = false

~
~
~
~
~
~
~
~
~
~
~
~
```

Flags 推荐

--single-threaded：禁用后台线程
--jitless：禁用jit代码的生成

使用模拟器进行跟踪

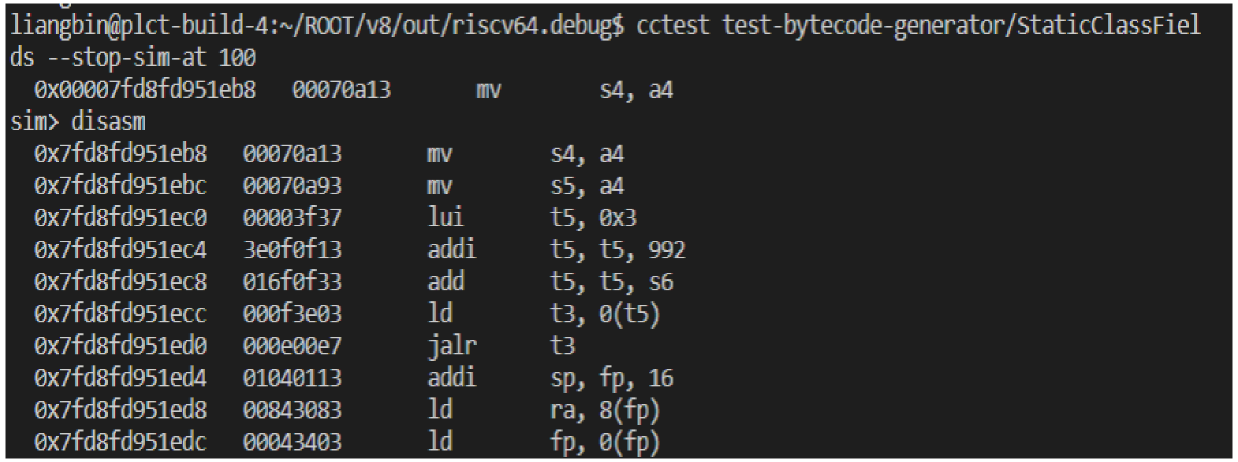
执行命令： cctest test-bytecode-generator/StaticClassFields --trace-sim > trace.log
生成trace.log文件：



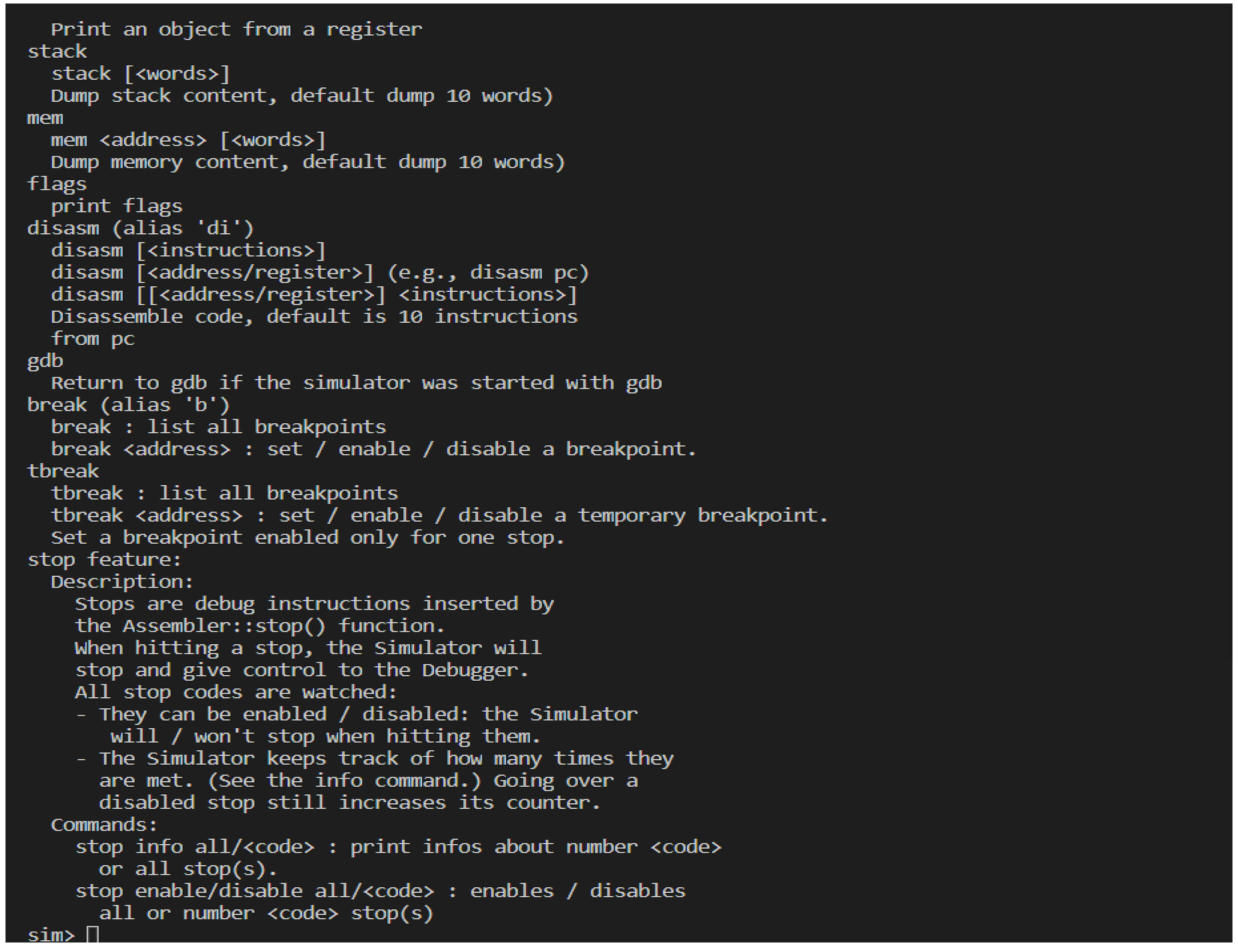
```
CallImpl: reg_arg_count = 6 entry-pc (JSEntry) = 0x7f1aa8d65800 a0 (Isolate) = 0x559f59a82720 a1 (orig_func/n
ew_target) = 0xfeb0e81599 a2 (func/target) = 0x527e35dc51 a3 (receiver) = 0x527e341119 a4 (argc) = 0x0 a5 (ar
gv) = 0x0
  0x7f1aa8d65800    f9810113       addi      sp, sp, -104         00007f1a9c72af68    (1)    int64:1397522706
38952 uint64:139752270638952
  0x7f1aa8d65804    06113023       sd        ra, 96(sp)                               (2)    int64:-2 uint64:
18446744073709551614 --> [addr: 7f1a9c72afc8]
  0x7f1aa8d65808    04813c23       sd        fp, 88(sp)                               (3)    int64:0 uint64:0
 --> [addr: 7f1a9c72afc0]
  0x7f1aa8d6580c    05b13823       sd        s11, 80(sp)                              (4)    int64:0 uint64:0
 --> [addr: 7f1a9c72afb8]
  0x7f1aa8d65810    05a13423       sd        s10, 72(sp)                              (5)    int64:0 uint64:0
 --> [addr: 7f1a9c72afb0]
  0x7f1aa8d65814    05913023       sd        s9, 64(sp)                               (6)    int64:0 uint64:0
 --> [addr: 7f1a9c72afa8]
  0x7f1aa8d65818    03813c23       sd        s8, 56(sp)                               (7)    int64:0 uint64:0
 --> [addr: 7f1a9c72afa0]
  0x7f1aa8d6581c    03713823       sd        s7, 48(sp)                               (8)    int64:0 uint64:0
 --> [addr: 7f1a9c72af98]
  0x7f1aa8d65820    03613423       sd        s6, 40(sp)                               (9)    int64:0 uint64:0
 --> [addr: 7f1a9c72af90]
  0x7f1aa8d65824    03513023       sd        s5, 32(sp)                               (10)    int64:0 uint64:
0 --> [addr: 7f1a9c72af88]
  0x7f1aa8d65828    01413c23       sd        s4, 24(sp)                               (11)    int64:0 uint64:
0 --> [addr: 7f1a9c72af80]
  0x7f1aa8d6582c    01313823       sd        s3, 16(sp)                               (12)    int64:0 uint64:
0 --> [addr: 7f1a9c72af78]
  0x7f1aa8d65830    01213423       sd        s2, 8(sp)                                (13)    int64:0 uint64:
0 --> [addr: 7f1a9c72af70]
  0x7f1aa8d65834    00913023       sd        s1, 0(sp)                                (14)    int64:0 uint64:
0 --> [addr: 7f1a9c72af68]
  0x7f1aa8d65838    fa010113       addi      sp, sp, -96          00007f1a9c72af08    (15)    int64:139752270
638856 uint64:139752270638856
```

▶▶ 使用模拟器进行调试

执行命令： cctest test-bytecode-generator/StaticClassFields --stop-sim-at 100
在执行100条riscv指令后进入模拟器模式



```
liangbin@plct-build-4:~/ROOT/v8/out/riscv64.debug$ cctest test-bytecode-generator/StaticClassFiel
ds --stop-sim-at 100
  0x00007fd8fd951eb8   00070a13       mv       s4, a4
sim> disasm
  0x7fd8fd951eb8   00070a13       mv       s4, a4
  0x7fd8fd951ebc   00070a93       mv       s5, a4
  0x7fd8fd951ec0   00003f37       lui      t5, 0x3
  0x7fd8fd951ec4   3e0f0f13       addi     t5, t5, 992
  0x7fd8fd951ec8   016f0f33       add      t5, t5, s6
  0x7fd8fd951ecc   000f3e03       ld       t3, 0(t5)
  0x7fd8fd951ed0   000e00e7       jalr     t3
  0x7fd8fd951ed4   01040113       addi     sp, fp, 16
  0x7fd8fd951ed8   00843083       ld       ra, 8(fp)
  0x7fd8fd951edc   00043403       ld       fp, 0(fp)
```



```
  Print an object from a register
stack
  stack [<words>]
  Dump stack content, default dump 10 words)
mem
  mem <address> [<words>]
  Dump memory content, default dump 10 words)
flags
  print flags
disasm (alias 'di')
  disasm [<instructions>]
  disasm [<address/register>] (e.g., disasm pc)
  disasm [[<address/register>] <instructions>]
  Disassemble code, default is 10 instructions
  from pc
gdb
  Return to gdb if the simulator was started with gdb
break (alias 'b')
  break : list all breakpoints
  break <address> : set / enable / disable a breakpoint.
tbreak
  tbreak : list all breakpoints
  tbreak <address> : set / enable / disable a temporary breakpoint.
  Set a breakpoint enabled only for one stop.
stop feature:
  Description:
    Stops are debug instructions inserted by
    the Assembler::stop() function.
    When hitting a stop, the Simulator will
    stop and give control to the Debugger.
    All stop codes are watched:
    - They can be enabled / disabled: the Simulator
       will / won't stop when hitting them.
    - The Simulator keeps track of how many times they
      are met. (See the info command.) Going over a
      disabled stop still increases its counter.
  Commands:
    stop info all/<code> : print infos about number <code>
      or all stop(s).
    stop enable/disable all/<code> : enables / disables
      all or number <code> stop(s)
sim> 
```

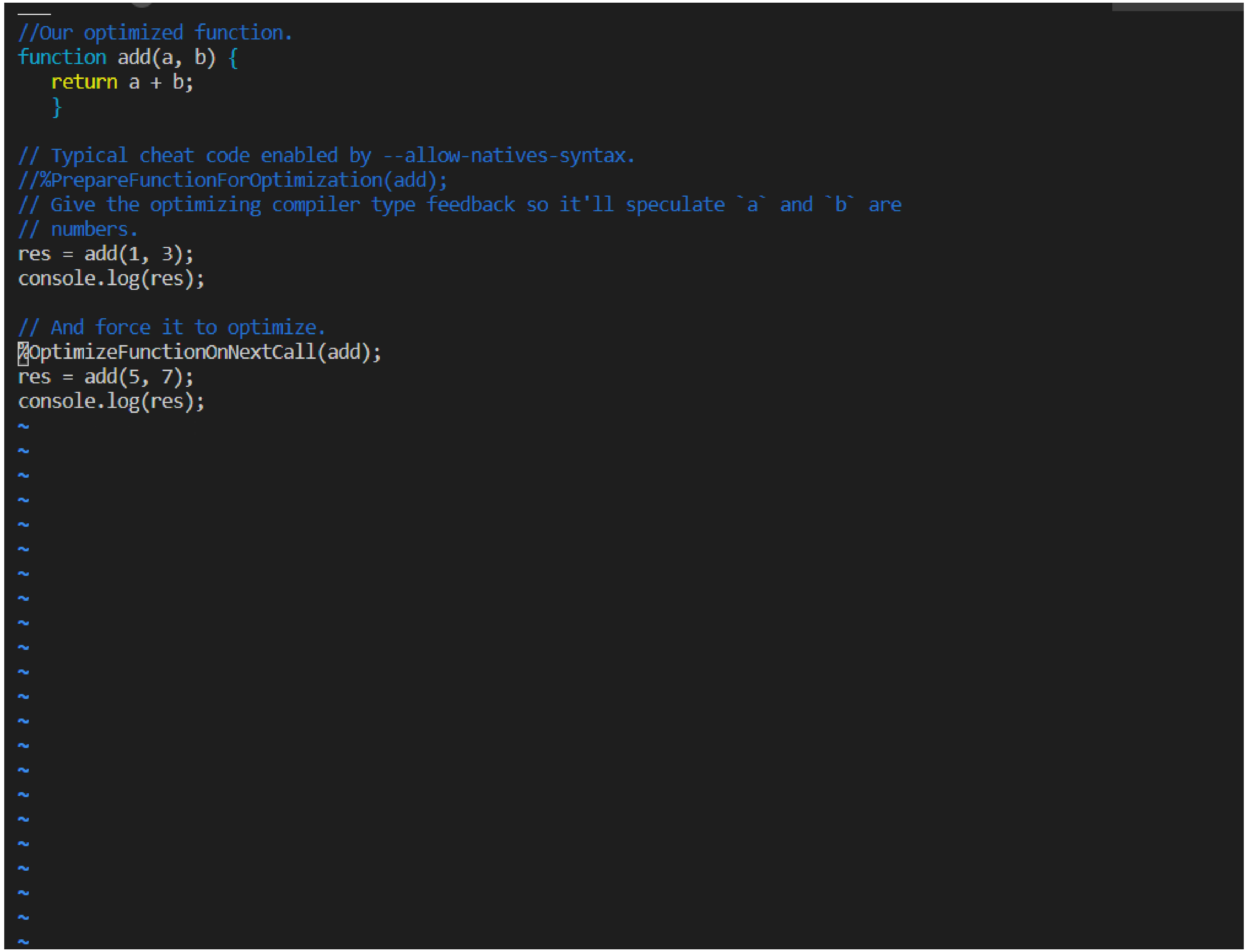
Help 介绍

```
cont (alias 'c')
  Continue execution
stepi (alias 'si')
  Step one instruction
print (alias 'p')
  print <register>
  Print register content
  Use register name 'all' to print all GPRs
  Use register name 'allf' to print all GPRs and FPRs
printobject (alias 'po')
  printobject <register>
  Print an object from a register
stack
  stack [<words>]
  Dump stack content, default dump 10 words)
mem
  mem <address> [<words>]
  Dump memory content, default dump 10 words)
flags
  print flags
disasm (alias 'di')
  disasm [<instructions>]
  disasm [<address/register>] (e.g., disasm pc)
  disasm [[<address/register>] <instructions>]
  Disassemble code, default is 10 instructions
  from pc
gdb
  Return to gdb if the simulator was started with gdb
break (alias 'b')
  break : list all breakpoints
  break <address> : set / enable / disable a breakpoint.
tbreak
  tbreak : list all breakpoints
  tbreak <address> : set / enable / disable a temporary breakpoint.
  Set a breakpoint enabled only for one stop.
```

调试 APIs

Trap() (处理信号）
Assert(Condition cc, AbortReason reason, Register rs, Operand rt)（断言，进行报错）
Check(Condition cc, AbortReason reason, Register rs, Operand rt)（检查约束）
Abort(AbortReason reason)（终止进程）
stop(uint32_t code = kMaxStopCode)（停止进程）
break_(uint32_t code, bool break_as_stop = false)（中断进程）

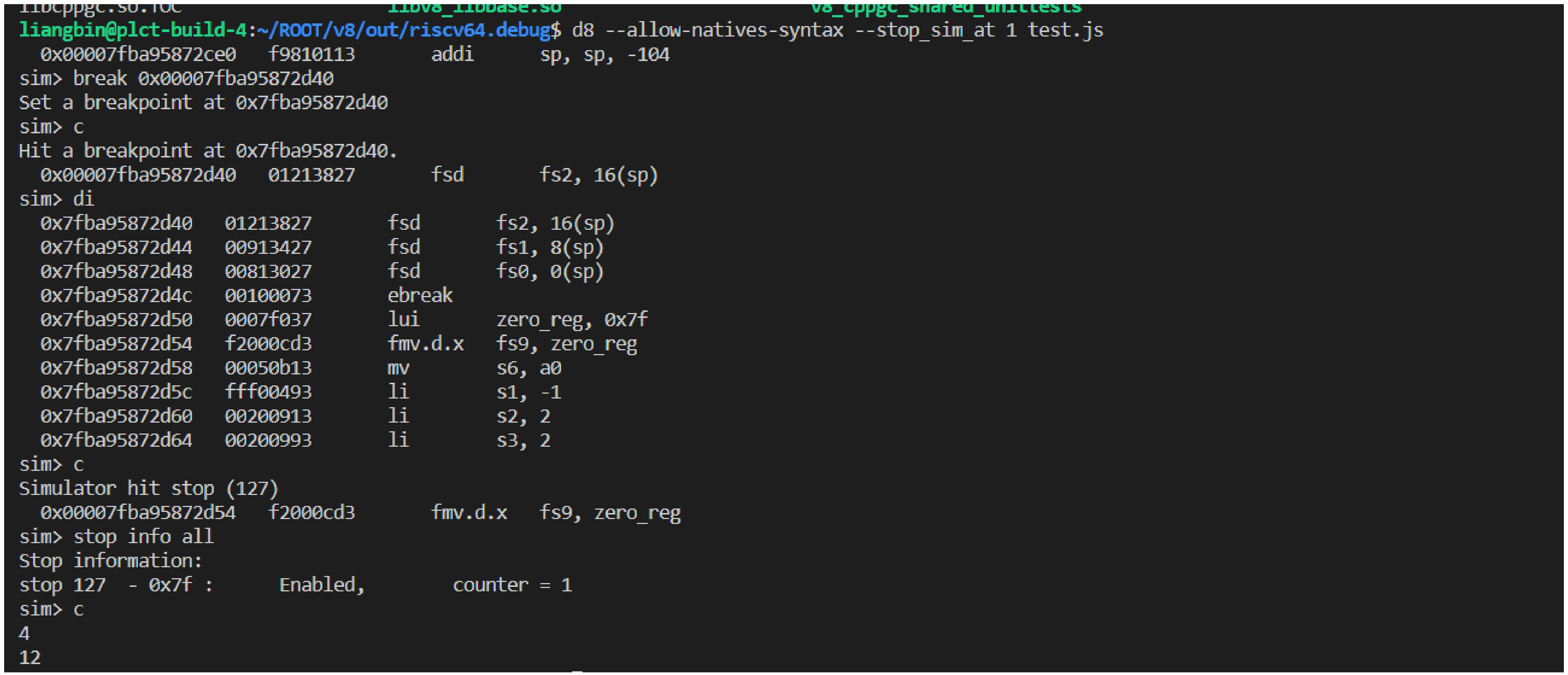
测试样例



```
//Our optimized function.
function add(a, b) {
   return a + b;
   }

// Typical cheat code enabled by --allow-natives-syntax.
//%PrepareFunctionForOptimization(add);
// Give the optimizing compiler type feedback so it'll speculate `a` and `b` are
// numbers.
res = add(1, 3);
console.log(res);

// And force it to optimize.
%OptimizeFunctionOnNextCall(add);
res = add(5, 7);
console.log(res);
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

代码注释

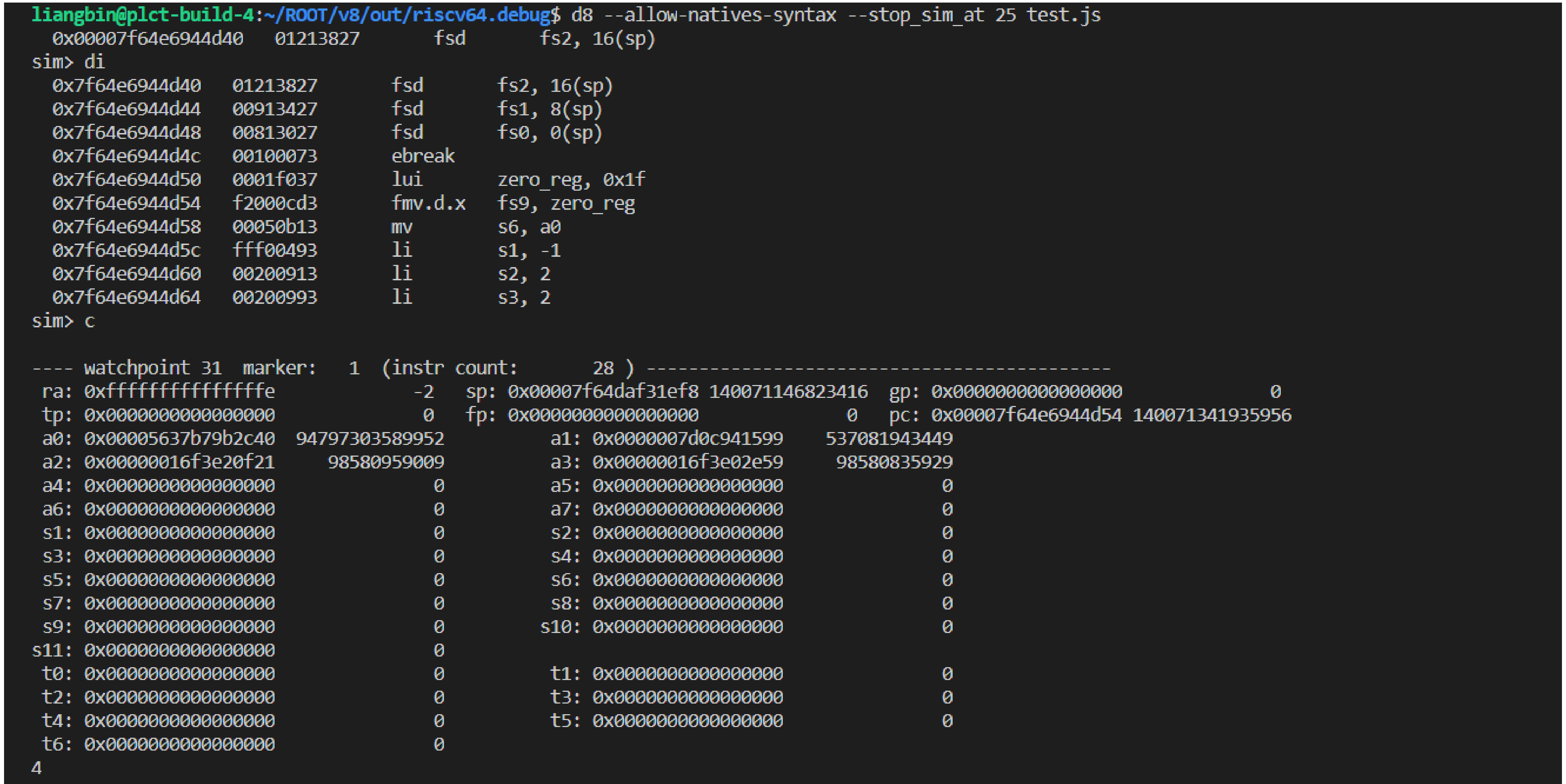
在macro-assembler-riscv64.cc:2183前加入stop（127）
执行命令： out/riscv64.sim/d8 --allow-natives-syntax --stop_sim_at 1 ../test.js



```
liangbin@plct-build-4:~/ROOT/v8/out/riscv64.debug$ d8 --allow-natives-syntax --stop_sim_at 1 test.js
  0x00007fba95872ce0    f9810113       addi      sp, sp, -104
sim> break 0x00007fba95872d40
Set a breakpoint at 0x7fba95872d40
sim> c
Hit a breakpoint at 0x7fba95872d40.
  0x00007fba95872d40    01213827       fsd       fs2, 16(sp)
sim> di
  0x7fba95872d40    01213827       fsd       fs2, 16(sp)
  0x7fba95872d44    00913427       fsd       fs1, 8(sp)
  0x7fba95872d48    00813027       fsd       fs0, 0(sp)
  0x7fba95872d4c    00100073       ebreak
  0x7fba95872d50    0007f037       lui       zero_reg, 0x7f
  0x7fba95872d54    f2000cd3       fmv.d.x   fs9, zero_reg
  0x7fba95872d58    00050b13       mv        s6, a0
  0x7fba95872d5c    fff00493       li        s1, -1
  0x7fba95872d60    00200913       li        s2, 2
  0x7fba95872d64    00200993       li        s3, 2
sim> c
Simulator hit stop (127)
  0x00007fba95872d54    f2000cd3       fmv.d.x   fs9, zero_reg
sim> stop info all
Stop information:
stop 127  - 0x7f :      Enabled,        counter = 1
sim> c
4
12
```

▶▶ 测试样例

在macro-assembler-riscv64.cc:2183前加入break_(31, false);
执行命令：out/riscv64.sim/d8 --allow-natives-syntax --stop_sim_at 25 ../test.js



```
liangbin@plct-build-4:~/ROOT/v8/out/riscv64.debug$ d8 --allow-natives-syntax --stop_sim_at 25 test.js
  0x00007f64e6944d40   01213827       fsd       fs2, 16(sp)
sim> di
  0x7f64e6944d40   01213827       fsd       fs2, 16(sp)
  0x7f64e6944d44   00913427       fsd       fs1, 8(sp)
  0x7f64e6944d48   00813027       fsd       fs0, 0(sp)
  0x7f64e6944d4c   00100073       ebreak
  0x7f64e6944d50   0001f037       lui       zero_reg, 0x1f
  0x7f64e6944d54   f2000cd3       fmv.d.x   fs9, zero_reg
  0x7f64e6944d58   00050b13       mv        s6, a0
  0x7f64e6944d5c   fff00493       li        s1, -1
  0x7f64e6944d60   00200913       li        s2, 2
  0x7f64e6944d64   00200993       li        s3, 2
sim> c

---- watchpoint 31  marker:   1  (instr count:       28 ) --------------------------------------------
 ra: 0xfffffffffffffffe             -2   sp: 0x00007f64daf31ef8 140071146823416  gp: 0x0000000000000000              0
 tp: 0x0000000000000000              0   fp: 0x0000000000000000               0   pc: 0x00007f64e6944d54 140071341935956
 a0: 0x00005637b79b2c40  94797303589952          a1: 0x0000007d0c941599    537081943449
 a2: 0x00000016f3e20f21     98580959009          a3: 0x00000016f3e02e59     98580835929
 a4: 0x0000000000000000               0          a5: 0x0000000000000000               0
 a6: 0x0000000000000000               0          a7: 0x0000000000000000               0
 s1: 0x0000000000000000               0          s2: 0x0000000000000000               0
 s3: 0x0000000000000000               0          s4: 0x0000000000000000               0
 s5: 0x0000000000000000               0          s6: 0x0000000000000000               0
 s7: 0x0000000000000000               0          s8: 0x0000000000000000               0
 s9: 0x0000000000000000               0         s10: 0x0000000000000000               0
s11: 0x0000000000000000               0
 t0: 0x0000000000000000               0          t1: 0x0000000000000000               0
 t2: 0x0000000000000000               0          t3: 0x0000000000000000               0
 t4: 0x0000000000000000               0          t5: 0x0000000000000000               0
 t6: 0x0000000000000000               0
4
```

将模拟的跟踪连接到代码

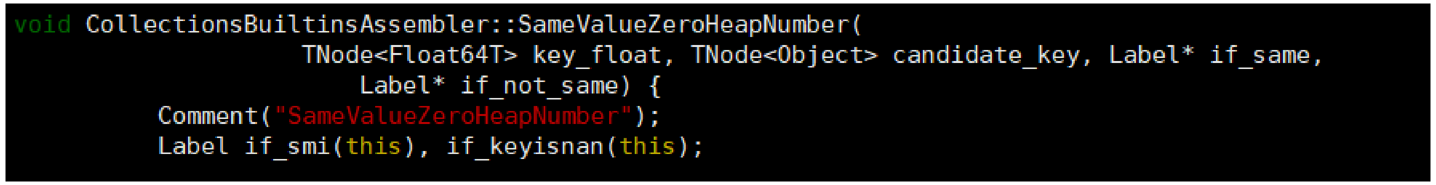
--print-bytecode： print Ignition interpreter bytecode
--print-code and --code-comments: print out generated final code (w/ additional comments)
--print-all-code: print out all generated codes
--use-verbose-printer:print out
--print-wasm-code, --print-wasm-stub-code, and --code-coments: printout wasm code generated and wasm native code gen
--print-builtin-code:print out disasm of generated builtin functions
--print-builtin-size:print out their instruction sizes

命令示例：

```
cctest test-bytecode-generator/StaticClassFields --print-all-code > t.all-code
```

⏩ 代码标签

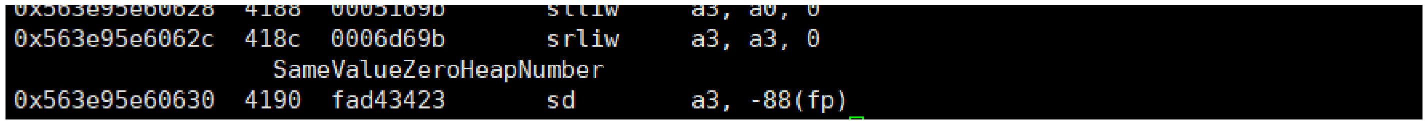
插入comments：



```
void CollectionsBuiltinsAssembler::SameValueZeroHeapNumber(
                    TNode<Float64T> key_float, TNode<Object> candidate_key, Label* if_same,
                        Label* if_not_same) {
          Comment("SameValueZeroHeapNumber");
          Label if_smi(this), if_keyisnan(this);
```

前提条件，在args中加入参数:v8_enable_snapshot_code_comments = true

结果：



```
0x563e95e60628  4188  0005169b       slliw     a3, a0, 0
0x563e95e6062c  418c  0006d69b       srliw     a3, a3, 0
                  SameValueZeroHeapNumber
0x563e95e60630  4190  fad43423       sd        a3, -88(fp)
```

可用工具

有多种工具可用于帮助您进行v8的debug
analyze.py：
可从v8-riscv-tools中获得
被设计用于读取标志模拟输出--print-all-code和--trace-sim，并写入文件。
具体用法：

```
$ cctest --print-all-code -trace-sim test-interpreter-intrinsics/Call &> out
$ analyze.py out
```

Turbolize：
可产生存储IR/code的json文件。
用法：
在执行编译时加入：--trace-turbo/--trace-turbo-path=xxx

countInstr.py
一个简单的工具，可以收集统计数据并比较两个不同后端生成的代码。
用法：

```
python3 ./v8-riscv-tools/CountInstr.py path_to_riscv_d8 path_to_mips_d8  [args of d8]
```

THANKS!

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa.