```
//对应文件: src\cpu\x86\vm\templateInterpreter_x86_32.cpp
//对应方法: 1401行的InterpreterGenerator::generate_normal_entry(bool synchronized)

method entry point (kind = zerolocals)  [0x07f0ba60, 0x07f0bde0]  896 bytes

  0x07f0ba60: mov    0x8(%ebx),%edx //执行call_stub后ebx指向method的地址，所以这条指令把ConstMethod的地址放到edx中

//ConstMethod的内存布局
/*
        偏移(10) 偏移(16)  字段                  类型
        ----     ------    --------        --------------------
        0           0         _fingerprint           volatile unsigned __int64，占了 8 个字节
    8        8          _constants             ConstantPool *
        12          C         _stackmap_data         Array<unsigned char> *
        16          10        _constMethod_size      int
        20          14        _flags                 unsigned short
        22          16        _code_size             unsigned short
        24          18        _name_index            unsigned short
        26          1A        _signature_index       unsigned short
        28          1C        _method_idnum          unsigned short
        30          1E        _max_stack             unsigned short
        32          20        _max_locals            unsigned short
        34          22        _size_of_parameters    unsigned short
*/
  0x07f0ba63: movzwl 0x22(%edx),%ecx //_size_of_parameters
  0x07f0ba67: movzwl 0x20(%edx),%edx //_max_locals
  0x07f0ba6b: sub    %ecx,%edx //_max_locals - _size_of_parameters，相当于除了方法参数以外最大的局部变量slot是多少

  //----------begin void InterpreterGenerator::generate_stack_overflow_check(void)----------
          0x07f0ba6d: cmp    $0x3f6,%edx //检查_max_locals - _size_of_parameters是否超过$0x3f6
          0x07f0ba73: jbe    0x07f0baf3

          0x07f0ba79: push   %esi

          //_ get_thread(thread);
          0x07f0ba7a: mov    %fs:0x0(,%eiz,1),%esi
          0x07f0ba82: mov    -0xc(%esi),%esi
          // locals + overhead, in bytes
          0x07f0ba85: lea    0x28(,%edx,4),%eax
          // verify that thread stack base is non-zero
          0x07f0ba8c: cmpl   $0x0,0xd4(%esi)
          0x07f0ba96: jne    0x07f0baad
          //__ stop("stack base is zero");
          0x07f0ba9c: push   $0x553191c4
          0x07f0baa1: call   0x07f0baa6
          0x07f0baa6: pusha
          0x07f0baa7: call   0x54dedbf0
          0x07f0baac: hlt
          // verify that thread stack size is non-zero
          0x07f0baad: cmpl   $0x0,0xd8(%esi)
          0x07f0bab7: jne    0x07f0bace
          //__ stop("stack size is zero");
          0x07f0babd: push   $0x553191d8
          0x07f0bac2: call   0x07f0bac7
          0x07f0bac7: pusha
          0x07f0bac8: call   0x54dedbf0
          0x07f0bacd: hlt
          // Add stack base to locals and subtract stack size
          0x07f0bace: add    0xd4(%esi),%eax
          0x07f0bad4: sub    0xd8(%esi),%eax
          // Use the maximum number of pages we might bang.
          0x07f0bada: add    $0x9000,%eax
          // check against the current stack bottom
          0x07f0bae0: cmp    %eax,%esp
          0x07f0bae2: ja     0x07f0baf2
          // get saved bcp / (c++ prev state ).
          0x07f0bae8: pop    %esi
          // return address must be moved if SP is changed
          0x07f0bae9: pop    %eax
          0x07f0baea: mov    %esi,%esp
          0x07f0baec: push   %eax
          // Use the shared runtime version of the StackOverflowError.
          0x07f0baed: jmp    0x01cb2890
          0x07f0baf2: pop    %esi
  //----------end   void InterpreterGenerator::generate_stack_overflow_check(void)----------

  // get return address
  0x07f0baf3: pop    %eax
  // compute beginning of parameters (rdi)
  0x07f0baf4: lea    -0x4(%esp,%ecx,4),%edi //让edi指向第一个参数在堆栈中的位置
  // rdx - # of additional locals
```

```
    // allocate space for locals
    // explicitly initialize locals
    0x07f0baf8: test   %edx,%edx
    0x07f0bafa: jle    0x01cbbb08
    0x01cbbb00: push   $0x0
    0x01cbbb05: dec    %edx
    0x01cbbb06: jg     0x01cbbb00

    //执行完上面 5 条指令和下面的那条push   %eax后的堆栈如下:
            //    [ return_from_Java     ] <--- rsp
            //    [ 0                    ] 总共_max_locals - _size_of_parameters个0
            //      ...
            //    [ 0                    ]
            //    [ 0                    ]
            //    [ argument word n      ]
            //      ...
            // -N [ argument word 1      ]
            // -7 [ Possible padding for stack alignment ]
            // -6 [ Possible padding for stack alignment ]
            // -5 [ Possible padding for stack alignment ]
            // -4 [ mxcsr save           ] <--- rsp_after_call
            // -3 [ saved rbx,            ]
            // -2 [ saved rsi            ]
            // -1 [ saved rdi            ]
            //  0 [ saved rbp,            ] <--- rbp,
            //  1 [ return address       ]
            //  2 [ ptr. to call wrapper ]
            //  3 [ result               ]
            //  4 [ result_type          ]
            //  5 [ method               ]
            //  6 [ entry_point          ]
            //  7 [ parameters           ]
            //  8 [ parameter_size       ]
            //  9 [ thread               ]

    //----------begin void TemplateInterpreterGenerator::generate_fixed_frame(bool native_call)----------
            0x01cbbb08: push   %eax
            //__ enter();
            0x01cbbb09: push   %ebp
            0x01cbbb0a: mov    %esp,%ebp
            // set sender sp
            0x01cbbb0c: push   %esi //最后一个参数在堆栈中的位置
            0x01cbbb0d: push   $0x0
            0x01cbbb12: mov    0x8(%ebx),%esi  //ebx指向method, 0x8(%ebx)指向ConstMethod
            //0x28是40，按8字节对齐，在_size_of_parameters后有 4 字节全是 0 (用来补齐的)，
            //在这一步让%esi指向codebase，也就是method的第一个字节码的位置
            0x01cbbb15: lea    0x28(%esi),%esi
            0x01cbbb18: push   %ebx
            0x01cbbb19: push   $0x0
            0x01cbbb1e: mov    0x8(%ebx),%edx //ConstMethod *
            0x01cbbb21: mov    0x8(%edx),%edx //ConstantPool *
            0x01cbbb24: mov    0xc(%edx),%edx //ConstantPoolCache *
            // set constant pool cache
            0x01cbbb27: push   %edx
            // set locals pointer
            0x01cbbb28: push   %edi
            // set bcp
            0x01cbbb29: push   %esi
            // reserve word for pointer to expression stack bottom
            0x01cbbb2a: push   $0x0
            // set expression stack bottom
            0x01cbbb2f: mov    %esp,(%esp)
    //----------end   void TemplateInterpreterGenerator::generate_fixed_frame(bool native_call)----------

    // make sure method is not native & not abstract
    0x01cbbb32: mov    0x14(%ebx),%eax //_access_flags

    0x01cbbb35: test   $0x100,%eax //_access_flags等于JVM_ACC_NATIVE(0x0100，在jvm.h中定义)
    0x01cbbb3a: je     0x01cbbb51
    //__ stop("tried to execute native method as non-native");
    0x01cbbb40: push   $0x55318b94
    0x01cbbb45: call   0x01cbbb4a
    0x01cbbb4a: pusha
    0x01cbbb4b: call   0x54dedbf0
    0x01cbbb50: hlt

    0x01cbbb51: test   $0x400,%eax //_access_flags等于JVM_ACC_ABSTRACT(0x0400，在jvm.h中定义)
    0x01cbbb56: je     0x01cbbb6d
    //__ stop("tried to execute abstract method in interpreter");
    0x01cbbb5c: push   $0x55318bc4
    0x01cbbb61: call   0x01cbbb66
```

```
    0x01cbbb66: pusha
    0x01cbbb67: call   0x54dedbf0
    0x01cbbb6c: hlt

    // __ get_thread(rax);
    0x01cbbb6d: mov    %fs:0x0(,%eiz,1),%eax
    0x01cbbb75: mov    -0xc(%eax),%eax //得到thread指针 //为什么要减12, 见os::os_exception_wrapper
    //__ movbool(do_not_unlock_if_synchronized, true);
    0x01cbbb78: movb   $0x1,0x1a1(%eax) //把thread对象的_do_not_unlock_if_synchronized字段设为true


            //执行完上面的汇编代码后，此时的堆栈如下
                    //    [ (%esp)                ] <--- rsp  //reserve word for pointer to expression stack bottom
                    //    [ 第一个字节码内存地址 ]
                    //    [ argument word 1 所在堆栈位置 ]
                    //    [ ConstantPoolCache     ]
                    //    [ 0                     ]
                    //    [ method                ]
                    //    [ 0                     ]
                    //    [ argument word n 所在堆栈位置 ]
                    //    [ saved rbp,            ] <--- rbp,

                    //    [ return_from_Java      ]
                    //    [ 0                     ] 总共_max_locals - _size_of_parameters个0
                    //      ...
                    //    [ 0                     ]
                    //    [ 0                     ]
                    //    [ argument word n       ]
                    //      ...
                    // -N [ argument word 1       ]
                    // -7 [ Possible padding for stack alignment ]
                    // -6 [ Possible padding for stack alignment ]
                    // -5 [ Possible padding for stack alignment ]
                    // -4 [ mxcsr save            ] <--- rsp_after_call
                    // -3 [ saved rbx,            ]
                    // -2 [ saved rsi             ]
                    // -1 [ saved rdi             ]
                    //  0 [ saved rbp,            ] <--- rbp,
                    //  1 [ return address        ]
                    //  2 [ ptr. to call wrapper  ]
                    //  3 [ result                ]
                    //  4 [ result_type           ]
                    //  5 [ method                ]
                    //  6 [ entry_point           ]
                    //  7 [ parameters            ]
                    //  8 [ parameter_size        ]
                    //  9 [ thread                ]

    //---begin generate_counter_incr
            //---begin get_method_counters(rbx, rax, done);
                    0x01cbbb7f: mov    0x10(%ebx),%eax //MethodCounters指针
                    0x01cbbb82: test   %eax,%eax
                    //如果MethodCounters指针不为null，
                    //则直接跳过对InterpreterRuntime::build_method_counters方法的调用，
                    //在InterpreterRuntime::build_method_counters方法中会生成一个MethodCounters，
                    //然后赋值给metho._method_counters字段
                    0x01cbbb84: jne    0x01cbbc33 //ZF=0时才跳转, 也就是当%eax不为 0 时，%eax不为 0 就表示MethodCounters指针不为null
                    //---begin call_VM
                            0x01cbbb8a: call   0x01cbbb94
                            0x01cbbb8f: jmp    0x01cbbc28

                            //为调用InterpreterRuntime::build_method_counters(JavaThread* thread, Method* m)
                            //先把method指针压入堆栈
                            //pass_arg1(this, arg_1);
                            0x01cbbb94: push   %ebx,

                            //---begin call_VM_helper
                                    0x01cbbb95: lea    0x8(%esp),%eax //reserve word for pointer to expression stack bottom对应的
那个堆栈项的地址
                                    //---begin call_VM_base
                                            0x01cbbb99: cmpl   $0x0,-0x8(%ebp)
                                            0x01cbbba0: je     0x01cbbbb7
                                            //stop("InterpreterMacroAssembler::call_VM_base: last_sp != NULL");
                                            0x01cbbba6: push   $0x55310188
                                            0x01cbbbab: call   0x01cbbbb0
                                            0x01cbbbb0: pusha
                                            0x01cbbbb1: call   0x54dedbf0
                                            0x01cbbbb6: hlt

                                            //save_bcp()
                                            0x01cbbbb7: mov    %esi,-0x1c(%ebp) //放到[ 第一个字节码内存地址 ]那个堆栈项，其实跟%esi
```

中的值是一样

```
                                        //---begin MacroAssembler::call_VM_base
                                                //get_thread(java_thread);
                                                0x01cbbbba: mov    %fs:0x0(,%eiz,1),%edi
                                                0x01cbbbc2: mov    -0xc(%edi),%edi
                                                //NOT_LP64(push(java_thread); number_of_arguments++);
                                                0x01cbbbc5: push   %edi
                                                //set_last_Java_frame => if (last_java_fp->is_valid())
                                                0x01cbbbc6: mov    %ebp,0x144(%edi) //对应((thread)->_anchor)._last_Java_fp
                                                0x01cbbbcc: mov    %eax,0x13c(%edi) //对应((thread)->_anchor)._last_Java_sp
                                                //---begin MacroAssembler::call_VM_leaf_base
                                                    //调用InterpreterRuntime::build_method_counters(JavaThread* thread,
Method* m)
                                                        //此时堆栈栈顶的两项刚好是thread和method
                                                        0x01cbbbd2: call   0x5505d720
                                                        0x01cbbbd7: add    $0x8,%esp
                                                //---end   MacroAssembler::call_VM_leaf_base

                                                //guarantee(java_thread != rax, "change this code");
                                                0x01cbbbda: push   %eax
                                                //get_thread(java_thread);
                                                0x01cbbbdb: mov    %fs:0x0(,%eiz,1),%eax
                                                0x01cbbbe3: mov    -0xc(%eax),%eax

                                                0x01cbbbe6: cmp    %eax,%edi
                                                0x01cbbbe8: je     0x01cbbbff

                                                //STOP("MacroAssembler::call_VM_base: rdi not callee saved?");
                                                ;; MacroAssembler::call_VM_base: rdi not callee saved?
                                                0x01cbbbee: push   $0x55312af0
                                                0x01cbbbf3: call   0x01cbbbf8
                                                0x01cbbbf8: pusha
                                                0x01cbbbf9: call   0x54dedbf0
                                                0x01cbbbfe: hlt

                                                //pop(rax);
                                                0x01cbbbff: pop    %eax
                                                //reset_last_Java_frame(java_thread, true, false);
                                                0x01cbbc00: movl   $0x0,0x13c(%edi) //对应((thread)->_anchor)._last_Java_sp
                                                0x01cbbc0a: movl   $0x0,0x144(%edi) //对应((thread)->_anchor)._last_Java_fp
                                                // check for pending exceptions (java_thread is set upon return)
                                                0x01cbbc14: cmpl   $0x0,0x4(%edi) //对应_pending_exception
                                                0x01cbbc1b: jne    0x01cb0340
                                        //---end   MacroAssembler::call_VM_base

                                        //restore_bcp();
                                        0x01cbbc21: mov    -0x1c(%ebp),%esi
                                        //restore_locals();
                                        0x01cbbc24: mov    -0x18(%ebp),%edi
                                //---end   call_VM_base
                        //---end   call_VM_helper

                        0x01cbbc27: ret
                //---end   call_VM
                0x01cbbc28: mov    0x10(%ebx),%eax
                0x01cbbc2b: test   %eax,%eax //如查MethodCounters指针还是null，那么就跳过计数更新
                0x01cbbc2d: je     0x01cbbc50
        //---end   get_method_counters(rbx, rax, done);

        //MethodCounters的内存布局
        /*
                     偏移  字段                          类型
                     ----  ---------------               --------------------
                     0         _interpreter_invocation_count  int
                     4         _interpreter_throwout_count    unsigned short
                     6         _number_of_breakpoints         unsigned short
                     8         _invocation_counter._counter   unsigned int
                     12        _backedge_counter._counter     unsigned int
      */
        // Update standard invocation counters
        0x01cbbc33: mov    0x8(%eax),%ecx //_invocation_counter._counter
        0x01cbbc36: add    $0x8,%ecx
        0x01cbbc39: mov    %ecx,0x8(%eax)

        // load backedge counter
        0x01cbbc3c: mov    0xc(%eax),%eax //_backedge_counter._counter
        // mask out the status bits
        //InvocationCounter的格式
        // bit no: |31  3|  2  | 1 0 |
      // format: [count|carry|state]
        //所以要屏蔽掉后三位carry|state
```

```
        0x01cbbc3f: and    $0xfffffff8,%eax
        // add both counters
        //累计_invocation_counter+_backedge_counter,
        //注意这里只是更新ecx，并不会写回_invocation_counter
        0x01cbbc42: add    %eax,%ecx

    //是否超过InvocationCounter::InterpreterInvocationLimit，超过的话就当overflow处理
        0x01cbbc44: cmp    0x55627784,%ecx
        0x01cbbc4a: jae    0x07f0bd29
//---end   generate_counter_incr

// bang_stack_shadow_pages(false); 并且StackShadowPages=9
0x01cbbc50: mov    %eax,-0x1000(%esp)
0x01cbbc57: mov    %eax,-0x2000(%esp)
0x01cbbc5e: mov    %eax,-0x3000(%esp)
0x01cbbc65: mov    %eax,-0x4000(%esp)
0x01cbbc6c: mov    %eax,-0x5000(%esp)
0x01cbbc73: mov    %eax,-0x6000(%esp)
0x01cbbc7a: mov    %eax,-0x7000(%esp)
0x01cbbc81: mov    %eax,-0x8000(%esp)
0x01cbbc88: mov    %eax,-0x9000(%esp)

//__ get_thread(rax);
0x01cbbc8f: mov    %fs:0x0(,%eiz,1),%eax
0x01cbbc97: mov    -0xc(%eax),%eax
//__ movbool(do_not_unlock_if_synchronized, false);
0x01cbbc9a: movb   $0x0,0x1a1(%eax)

//对应zerolocals_synchronized中的lock_method
        // no synchronization necessary
        0x01cbbca1: mov    0x14(%ebx),%eax
        0x01cbbca4: test   $0x20,%eax
        0x01cbbca9: je     0x01cbbcc0
        //__ stop("method needs synchronization");
        0x01cbbcaf: push   $0x55318bf4
        0x01cbbcb4: call   0x01cbbcb9
        0x01cbbcb9: pusha
        0x01cbbcba: call   0x54dedbf0
        0x01cbbcbf: hlt

// start execution *****************非常重要*****************
0x01cbbcc0: mov    -0x20(%ebp),%eax //看看[ (%esp)               ]堆栈项中的值与esp是否一样
0x01cbbcc3: cmp    %esp,%eax
0x01cbbcc5: je     0x01cbbcdc
//__ stop("broken stack frame setup in interpreter");
0x01cbbccb: push   $0x55318c14
0x01cbbcd0: call   0x01cbbcd5
0x01cbbcd5: pusha
0x01cbbcd6: call   0x54dedbf0
0x01cbbcdb: hlt

// jvmti support
//---begin InterpreterMacroAssembler::notify_method_entry()
        //SkipIfEqual skip_if(this, &DTraceMethodProbes, 0);
        0x01cbbcdc: cmpb   $0x0,0x55633e5f
        0x01cbbce3: je     0x07f0bd1f
        //get_thread(rcx);
        0x01cbbce9: mov    %fs:0x0(,%eiz,1),%ecx
        0x01cbbcf1: mov    -0xc(%ecx),%ecx
        //get_method(rbx);
        0x01cbbcf4: mov    -0xc(%ebp),%ebx
        0x01cbbcf7: push   %ebx
        0x01cbbcf8: push   %ecx
        //InterpreterMacroAssembler::call_VM_leaf_base
        0x01cbbcf9: cmpl   $0x0,-0x8(%ebp)
        0x07f0bd00: je     0x07f0bd17
        //stop("InterpreterMacroAssembler::call_VM_leaf_base: last_sp != NULL");
        0x07f0bd06: push   $0x55310148
        0x07f0bd0b: call   0x07f0bd10
        0x07f0bd10: pusha
        0x07f0bd11: call   0x54dedbf0
        0x07f0bd16: hlt
        //MacroAssembler::call_VM_leaf_base(address entry_point, int number_of_arguments)
        0x07f0bd17: call   0x552155e0
        0x07f0bd1c: add    $0x8,%esp
//---end   InterpreterMacroAssembler::notify_method_entry()

// __ dispatch_next(vtos);
//从这里跳转到main方法的第一个字节码对应的汇编
0x07f0bd1f: movzbl (%esi),%ebx //%esi的值是第一条字节码在内存中的地址，(%esi)就是第一条字节码
//在VS中打断点时按"22 jmp"找
```

```
    0x07f0bd22: jmp    *0x55629838(,%ebx,4) //dispatch_base(state, Interpreter::dispatch_table(state));


    //---begin generate_counter_overflow
            0x07f0bd29: mov    $0x0,%eax
            //---begin call_VM
                    0x07f0bd2e: call   0x07f0bd38
                    0x07f0bd33: jmp    0x07f0bdcc
                    0x07f0bd38: push   %eax
                    //---begin call_VM_helper
                            0x07f0bd39: lea    0x8(%esp),%eax
                            //---begin call_VM_base
                                    0x07f0bd3d: cmpl   $0x0,-0x8(%ebp)
                                    0x07f0bd44: je     0x07f0bd5b
                                    //stop("InterpreterMacroAssembler::call_VM_base: last_sp != NULL");
                                    0x07f0bd4a: push   $0x55310188
                                    0x07f0bd4f: call   0x07f0bd54
                                    0x07f0bd54: pusha
                                    0x07f0bd55: call   0x54dedbf0
                                    0x07f0bd5a: hlt

                                    //save_bcp();
                                    0x07f0bd5b: mov    %esi,-0x1c(%ebp)
                                    //---begin MacroAssembler::call_VM_base
                                            //get_thread(java_thread);
                                            0x07f0bd5e: mov    %fs:0x0(,%eiz,1),%edi
                                            0x07f0bd66: mov    -0xc(%edi),%edi

                                            0x07f0bd69: push   %edi

                                            //set_last_Java_frame(java_thread, last_java_sp, rbp, NULL);
                                            0x07f0bd6a: mov    %ebp,0x144(%edi)
                                            0x07f0bd70: mov    %eax,0x13c(%edi)

                                            // MacroAssembler::call_VM_leaf_base
                                            0x07f0bd76: call   0x5505ce70
                                            0x07f0bd7b: add    $0x8,%esp

                                            //guarantee(java_thread != rax, "change this code");
                                            0x07f0bd7e: push   %eax
                                            //get_thread(rax);
                                            0x07f0bd7f: mov    %fs:0x0(,%eiz,1),%eax
                                            0x07f0bd87: mov    -0xc(%eax),%eax

                                            0x07f0bd8a: cmp    %eax,%edi
                                            0x07f0bd8c: je     0x07f0bda3
                                            //STOP("MacroAssembler::call_VM_base: rdi not callee saved?");
                                            ;; MacroAssembler::call_VM_base: rdi not callee saved?
                                            0x07f0bd92: push   $0x55312af0
                                            0x07f0bd97: call   0x07f0bd9c
                                            0x07f0bd9c: pusha
                                            0x07f0bd9d: call   0x54dedbf0
                                            0x07f0bda2: hlt

                                            0x07f0bda3: pop    %eax
                                            //reset_last_Java_frame(java_thread, true, false);
                                            0x07f0bda4: movl   $0x0,0x13c(%edi)
                                            0x07f0bdae: movl   $0x0,0x144(%edi)
                                            //if (check_exceptions)
                                            0x07f0bdb8: cmpl   $0x0,0x4(%edi)
                                            0x07f0bdbf: jne    0x01cb0340
                                    //---end   MacroAssembler::call_VM_base

                                    //restore_bcp();
                                    0x07f0bdc5: mov    -0x1c(%ebp),%esi
                                    //restore_locals();
                                    0x07f0bdc8: mov    -0x18(%ebp),%edi
                            //---end   call_VM_base
                    //---end   call_VM_helper

                    0x07f0bdcb: ret
            //---end   call_VM

            // restore Method*
            0x07f0bdcc: mov    -0xc(%ebp),%ebx
            // Preserve invariant that rsi/rdi contain bcp/locals of sender frame
            // and jump to the interpreted entry.
            0x07f0bdcf: jmp    0x01cbbc50
    //---end   generate_counter_overflow
```

```
0x07f0bdd4: int3
0x07f0bdd5: int3
0x07f0bdd6: int3
0x07f0bdd7: int3
0x07f0bdd8: int3
0x07f0bdd9: int3
0x07f0bdda: int3
0x07f0bddb: int3
0x07f0bddc: int3
0x07f0bddd: int3
0x07f0bdde: int3
0x07f0bddf: int3
```

对应VS中的Intel汇编格式

```
        01D2BA60  mov         edx,dword ptr [ebx+8]
        01D2BA63  movzx       ecx,word ptr [edx+22h]
        01D2BA67  movzx       edx,word ptr [edx+20h]
        01D2BA6B  sub         edx,ecx
        01D2BA6D  cmp         edx,3F6h
        01D2BA73  jbe         01D2BAF3
        01D2BA79  push        esi
        01D2BA7A  mov         esi,dword ptr fs:[0]
        01D2BA82  mov         esi,dword ptr [esi-0Ch]
        01D2BA85  lea         eax,[edx*4+28h]
        01D2BA8C  cmp         dword ptr [esi+0D4h],0
        01D2BA96  jne         01D2BAAD
        01D2BA9C  push        57CE91C4h
        01D2BAA1  call        01D2BAA6
        01D2BAA6  pushad
        01D2BAA7  call        MacroAssembler::debug32 (577BDBF0h)
        01D2BAAC  hlt

        01D2BAAD  cmp         dword ptr [esi+0D8h],0
        01D2BAB7  jne         01D2BACE
        01D2BABD  push        57CE91D8h
        01D2BAC2  call        01D2BAC7
        01D2BAC7  pushad
        01D2BAC8  call        MacroAssembler::debug32 (577BDBF0h)
        01D2BACD  hlt
        01D2BACE  add         eax,dword ptr [esi+0D4h]
        01D2BAD4  sub         eax,dword ptr [esi+0D8h]
        01D2BADA  add         eax,9000h
        01D2BAE0  cmp         esp,eax
        01D2BAE2  ja          01D2BAF2
        01D2BAE8  pop         esi
        01D2BAE9  pop         eax
        01D2BAEA  mov         esp,esi
        01D2BAEC  push        eax
        01D2BAED  jmp         01D22890
        01D2BAF2  pop         esi

        01D2BAF3  pop         eax
        01D2BAF4  lea         edi,[esp+ecx*4-4]
        01D2BAF8  test        edx,edx
        01D2BAFA  jle         01D2BB08
        01D2BB00  push        0
        01D2BB05  dec         edx
        01D2BB06  jg          01D2BB00

        01D2BB08  push        eax
        01D2BB09  push        ebp
        01D2BB0A  mov         ebp,esp
        01D2BB0C  push        esi
        01D2BB0D  push        0
        01D2BB12  mov         esi,dword ptr [ebx+8]
        01D2BB15  lea         esi,[esi+28h]
        01D2BB18  push        ebx
        01D2BB19  push        0
        01D2BB1E  mov         edx,dword ptr [ebx+8]
        01D2BB21  mov         edx,dword ptr [edx+8]
        01D2BB24  mov         edx,dword ptr [edx+0Ch]
        01D2BB27  push        edx
        01D2BB28  push        edi
        01D2BB29  push        esi
        01D2BB2A  push        0
        01D2BB2F  mov         dword ptr [esp],esp
        01D2BB32  mov         eax,dword ptr [ebx+14h]
        01D2BB35  test        eax,100h
        01D2BB3A  je          01D2BB51
        01D2BB40  push        57CE8B94h
        01D2BB45  call        01D2BB4A
```

```
01D2BB4A  pushad
01D2BB4B  call        MacroAssembler::debug32 (577BDBF0h)
01D2BB50  hlt

01D2BB51  test        eax,400h
01D2BB56  je          01D2BB6D
01D2BB5C  push        57CE8BC4h
01D2BB61  call        01D2BB66
01D2BB66  pushad
01D2BB67  call        MacroAssembler::debug32 (577BDBF0h)
01D2BB6C  hlt
01D2BB6D  mov         eax,dword ptr fs:[0]
01D2BB75  mov         eax,dword ptr [eax-0Ch]
01D2BB78  mov         byte ptr [eax+1A1h],1
01D2BB7F  mov         eax,dword ptr [ebx+10h]
01D2BB82  test        eax,eax
01D2BB84  jne         01D2BC33

01D2BB8A  call        01D2BB94
01D2BB8F  jmp         01D2BC28
01D2BB94  push        ebx
01D2BB95  lea         eax,[esp+8]
01D2BB99  cmp         dword ptr [ebp-8],0
01D2BBA0  je          01D2BBB7
01D2BBA6  push        57CE0188h
01D2BBAB  call        01D2BBB0
01D2BBB0  pushad
01D2BBB1  call        MacroAssembler::debug32 (577BDBF0h)
01D2BBB6  hlt

01D2BBB7  mov         dword ptr [ebp-1Ch],esi
01D2BBBA  mov         edi,dword ptr fs:[0]
01D2BBC2  mov         edi,dword ptr [edi-0Ch]
01D2BBC5  push        edi
01D2BBC6  mov         dword ptr [edi+144h],ebp
01D2BBCC  mov         dword ptr [edi+13Ch],eax
01D2BBD2  call        InterpreterRuntime::build_method_counters (57A2D720h)
01D2BBD7  add         esp,8


01D2BBDA  push        eax
01D2BBDB  mov         eax,dword ptr fs:[0]
01D2BBE3  mov         eax,dword ptr [eax-0Ch]
01D2BBE6  cmp         edi,eax
01D2BBE8  je          01D2BBFF
01D2BBEE  push        57CE2AF0h
01D2BBF3  call        01D2BBF8
01D2BBF8  pushad
01D2BBF9  call        MacroAssembler::debug32 (577BDBF0h)
01D2BBFE  hlt

01D2BBFF  pop         eax
01D2BC00  mov         dword ptr [edi+13Ch],0
01D2BC0A  mov         dword ptr [edi+144h],0
01D2BC14  cmp         dword ptr [edi+4],0
01D2BC1B  jne         01D20340

01D2BC21  mov         esi,dword ptr [ebp-1Ch]
01D2BC24  mov         edi,dword ptr [ebp-18h]
01D2BC27  ret
01D2BC28  mov         eax,dword ptr [ebx+10h]
01D2BC2B  test        eax,eax
01D2BC2D  je          01D2BC50

01D2BC33  mov         ecx,dword ptr [eax+8]
01D2BC36  add         ecx,8
01D2BC39  mov         dword ptr [eax+8],ecx
01D2BC3C  mov         eax,dword ptr [eax+0Ch]
01D2BC3F  and         eax,0FFFFFFF8h

01D2BC42  add         ecx,eax
01D2BC44  cmp         ecx,dword ptr ds:[57FF7784h]
01D2BC4A  jae         01D2BD29

01D2BC50  mov         dword ptr [esp-1000h],eax
01D2BC57  mov         dword ptr [esp-2000h],eax
01D2BC5E  mov         dword ptr [esp-3000h],eax
01D2BC65  mov         dword ptr [esp-4000h],eax
01D2BC6C  mov         dword ptr [esp-5000h],eax
01D2BC73  mov         dword ptr [esp-6000h],eax
01D2BC7A  mov         dword ptr [esp-7000h],eax
```

```
01D2BC81  mov         dword ptr [esp-8000h],eax
01D2BC88  mov         dword ptr [esp-9000h],eax
01D2BC8F  mov         eax,dword ptr fs:[0]
01D2BC97  mov         eax,dword ptr [eax-0Ch]
01D2BC9A  mov         byte ptr [eax+1A1h],0

01D2BCA1  mov         eax,dword ptr [ebx+14h]
01D2BCA4  test        eax,20h
01D2BCA9  je          01D2BCC0
01D2BCAF  push        57CE8BF4h
01D2BCB4  call        01D2BCB9
01D2BCB9  pushad
01D2BCBA  call        MacroAssembler::debug32 (577BDBF0h)
01D2BCBF  hlt

01D2BCC0  mov         eax,dword ptr [ebp-20h]
01D2BCC3  cmp         eax,esp
01D2BCC5  je          01D2BCDC
01D2BCCB  push        57CE8C14h
01D2BCD0  call        01D2BCD5
01D2BCD5  pushad
01D2BCD6  call        MacroAssembler::debug32 (577BDBF0h)
01D2BCDB  hlt

01D2BCDC  cmp         byte ptr ds:[58003E5Fh],0
01D2BCE3  je          01D2BD1F
01D2BCE9  mov         ecx,dword ptr fs:[0]
01D2BCF1  mov         ecx,dword ptr [ecx-0Ch]
01D2BCF4  mov         ebx,dword ptr [ebp-0Ch]
01D2BCF7  push        ebx
01D2BCF8  push        ecx
01D2BCF9  cmp         dword ptr [ebp-8],0
01D2BD00  je          01D2BD17

01D2BD06  push        57CE0148h
01D2BD0B  call        01D2BD10
01D2BD10  pushad
01D2BD11  call        MacroAssembler::debug32 (577BDBF0h)
01D2BD16  hlt

01D2BD17  call        SharedRuntime::dtrace_method_entry (57BE55E0h)
01D2BD1C  add         esp,8

01D2BD1F  movzx       ebx,byte ptr [esi]
01D2BD22  jmp         dword ptr [ebx*4+57FF9838h]

01D2BD29  mov         eax,0
01D2BD2E  call        01D2BD38
01D2BD33  jmp         01D2BDCC
01D2BD38  push        eax
01D2BD39  lea         eax,[esp+8]
01D2BD3D  cmp         dword ptr [ebp-8],0
01D2BD44  je          01D2BD5B
01D2BD4A  push        57CE0188h
01D2BD4F  call        01D2BD54
01D2BD54  pushad
01D2BD55  call        MacroAssembler::debug32 (577BDBF0h)
01D2BD5A  hlt

01D2BD5B  mov         dword ptr [ebp-1Ch],esi
01D2BD5E  mov         edi,dword ptr fs:[0]
01D2BD66  mov         edi,dword ptr [edi-0Ch]
01D2BD69  push        edi
01D2BD6A  mov         dword ptr [edi+144h],ebp
01D2BD70  mov         dword ptr [edi+13Ch],eax
01D2BD76  call        InterpreterRuntime::frequency_counter_overflow (57A2CE70h)
01D2BD7B  add         esp,8

01D2BD7E  push        eax
01D2BD7F  mov         eax,dword ptr fs:[0]
01D2BD87  mov         eax,dword ptr [eax-0Ch]
01D2BD8A  cmp         edi,eax
01D2BD8C  je          01D2BDA3

01D2BD92  push        57CE2AF0h
01D2BD97  call        01D2BD9C
01D2BD9C  pushad
01D2BD9D  call        MacroAssembler::debug32 (577BDBF0h)
01D2BDA2  hlt

01D2BDA3  pop         eax
```

```
01D2BDA4  mov         dword ptr [edi+13Ch],0
01D2BDAE  mov         dword ptr [edi+144h],0
01D2BDB8  cmp         dword ptr [edi+4],0
01D2BDBF  jne         01D20340
01D2BDC5  mov         esi,dword ptr [ebp-1Ch]
01D2BDC8  mov         edi,dword ptr [ebp-18h]
01D2BDCB  ret
01D2BDCC  mov         ebx,dword ptr [ebp-0Ch]
01D2BDCF  jmp         01D2BC50
01D2BDD4  int         3
01D2BDD5  int         3
01D2BDD6  int         3
01D2BDD7  int         3
01D2BDD8  int         3
01D2BDD9  int         3
01D2BDDA  int         3
01D2BDDB  int         3
01D2BDDC  int         3
01D2BDDD  int         3
01D2BDDE  int         3
01D2BDDF  int         3
```