Root Cause Analysis of the in the wild JIT bug (CVE-2022-42856)

VoidiStaff

32 мин. на чтение

Посмотреть оригинал

iOS Security update for CVE-2022-42856

A few months ago, there was a security patch for an potentially actively exploited WebKit vulnerability in iOS released before iOS 15.1. The CVE number is **CVE-2022-42856**.

- iOS 16.1.2 (Released November 30, 2022) : https://support.apple.com/en-us/HT213516
- iOS 16.2 (Released December 13, 2022) : https://support.apple.com/en-us/HT213530
- iOS 15.7.2 (Released December 13, 2022) : https://support.apple.com/en-us/HT213531
- iOS 12.5.7 (Released January 23, 2023) : https://support.apple.com/en-us/HT213597

```
iOS 16.1.2
Released November 30, 2022
---
- WebKit

- Available for: iPhone 8 and later

- Impact: Processing maliciously crafted web content may lead to arbitrary code execution. Apple is aware of a report that this issue may have been actively exploited against versions of iOS released before iOS 15.1.

- Description: A type confusion issue was addressed with improved state handling.
---
WebKit Bugzilla: 248266
CVE-2022-42856: Clément Lecigne of Google's Threat Analysis Group
```

Patch

I searched for the bug id with the command `git log --all --grep="248266"` and found the related commit.

The provenType filtering in FTL's speculateRealNumber is incorrect.

commit 08cd8b07aa56b62ac7d11e241d884e3c5a1681df

Diff /Source/JavaScriptCore/ftl/FTLLowerDFGToB3.cpp

```
    void speculateRealNumber(Edge edge)
    {
        // Do an early return here because lowDouble() can create a lot of control flow.
        if (!m_interpreter.needsTypeCheck(edge))
            return;

        LValue value = lowJSValue(edge, ManualOperandSpeculation);
        LValue doubleValue = unboxDouble(value);

-       LBasicBlock intCase = m_out.newBlock();
+       LBasicBlock intOrNaNCase = m_out.newBlock();
        LBasicBlock continuation = m_out.newBlock();

        m_out.branch(
            m_out.doubleEqual(doubleValue, doubleValue),
-           usually(continuation), rarely(intCase));
+           usually(continuation), rarely(intOrNaNCase));

-       LBasicBlock lastNext = m_out.appendTo(intCase, continuation);
+       LBasicBlock lastNext = m_out.appendTo(intOrNaNCase, continuation);

        typeCheck(
            jsValueValue(value), m_node->child1(), SpecBytecodeRealNumber,
-           isNotInt32(value, provenType(m_node->child1()) & ~SpecFullDouble));
+           isNotInt32(value, provenType(m_node->child1()) & ~SpecDoubleReal));
        m_out.jump(continuation);

        m_out.appendTo(continuation, lastNext);
    }
```

Four lines have been changed, three of which are variable name modifications(`intCase` to `intOrNaNCase`), and the last line is the core of the vulnerability patch(`~SpecFullDouble` to `~SpecDoubleReal`).

Background

Speculative Compilation

JavaScript engine consists of a combination of interpreters and compilers. JavaScriptCore has a **4-layer compiler pipeline**. The code is initially executed by the interpreter, collecting profiling information, and code that has been executed enough times goes through a compilation layer according to the optimization level. Profiling information is continuously collected in each layer. Each layer optimizes code using **speculative compilation** mechanism based on profiling information. Speculative compilation make dynamic languages run faster.

```
-> Profiling
LLInt | Baseline JIT | DFG JIT | FTL JIT
<- OSR
```

- LLInt, low-level interpreter
  - It runs on the same stack as the JITs and uses a known set of registers and stack locations for its internal state.
- Baseline JIT, bytecode template JIT
  - It emits a template of machine code for each bytecode instruction without tring to reason about relationships between multiple instructions in the function. It compiles whole functions to method JIT. no OSR speculations but handful of diamond speculations based on profiling from the LLInt.
- DFG JIT, data flow graph JIT
  - OSR speculation based on profiling from the LLInt, Baseline. rare cases even using profiling data collected by the DFG JIT and FTL JIT. DFG IR allows for sophisticated reasoning about speculation. Avoid doing expensive optimizations and make many compromises to enable fast code generation.
- FTL JIT, faster than light JIT
  - Comprehensive compiler optimizations. Designed for peak throughput. Reuse most of the DFG JIT's optimizations and add lots more. The FTL JIT uses multiple IRs (DFG IR, DFG SSA IR, B3 IR, and Assembly IR).

If you want more knowledge, read [this article].

Root Cause Analysis

What is wrong?

```
void speculateRealNumber(Edge edge)
{
    // Do an early return here because lowDouble() can create a lot of control flow.
    if (!m_interpreter.needsTypeCheck(edge))
        return;

    LValue value = lowJSValue(edge, ManualOperandSpeculation);
    LValue doubleValue = unboxDouble(value);

    LBasicBlock intCase = m_out.newBlock();
    LBasicBlock continuation = m_out.newBlock();

    m_out.branch(
        m_out.doubleEqual(doubleValue, doubleValue),
        usually(continuation), rarely(intCase));

    LBasicBlock lastNext = m_out.appendTo(intCase, continuation);

    typeCheck(
        jsValueValue(value), m_node->child1(), SpecBytecodeRealNumber,
        isNotInt32(value, provenType(m_node->child1()) & ~SpecFullDouble));
    m_out.jump(continuation);

    m_out.appendTo(continuation, lastNext);
}
```

`doubleEqual` filters out double values which are not NaN. If `doubleValue` is a double value other than NaN, it goes to the continuation blcok, and if it is NaN, it goes to

intCase block.

SpecFullDouble

```
static constexpr SpeculatedType SpecAnyIntAsDouble                    = 1ull << 36; // It's
definitely an Int52 and it's inside a double.
static constexpr SpeculatedType SpecNonIntAsDouble                    = 1ull << 37; // It's
definitely not an Int52 but it's a real number and it's a double.
static constexpr SpeculatedType SpecDoubleReal                        = SpecNonIntAsDouble |
SpecAnyIntAsDouble; // It's definitely a non-NaN double.
...
static constexpr SpeculatedType SpecDoublePureNaN                     = 1ull << 38; // It's
definitely a NaN that is safe to tag (i.e. pure).
static constexpr SpeculatedType SpecDoubleImpureNaN                   = 1ull << 39; // It's
definitely a NaN that is unsafe to tag (i.e. impure).
static constexpr SpeculatedType SpecDoubleNaN                         = SpecDoublePureNaN |
SpecDoubleImpureNaN; // It's definitely some kind of NaN.
...
static constexpr SpeculatedType SpecFullDouble                        = SpecDoubleReal |
SpecDoubleNaN; // It's either a non-NaN or a NaN double.
```

SpecFullDouble is (1ull << 36 | 1ull << 37 | 1ull << 38 | 1ull << 39). It's means (SpecDoubleReal | SpecDoubleNaN). Same as (SpecNonIntAsDouble | SpecAnyIntAsDouble | SpecDoublePureNaN | SpecDoubleImpureNaN).

NaN is (1ull << 32 | 1ull << 33 | 1ull << 36 | 1ull << 37 | 1ull << 38). It's means (SpecInt32Only | SpecDoubleReal | SpecDoublePureNaN). Same as (**SpecBoolInt32** | **SpecNonBoolInt32** | SpecNonIntAsDouble | SpecAnyIntAsDouble | SpecDoublePureNaN).

If `m_node->child1()` is NaN, result of `provenType(m_node->child1()) & ~SpecFullDouble)` is (SpecBoolInt32 | SpecNonBoolInt32). It's **SpecInt32Only**. This is passed to **isNotInt32** function.

```
LValue isNotInt32(LValue jsValue, SpeculatedType type = SpecFullTop)
{
    if (LValue proven = isProvenValue(type, ~SpecInt32Only))
        return proven;
    return m_out.below(jsValue, m_numberTag);
}
```

In this function, **isProvenValue** take `SpecInt32Only` and `~SpecInt32Only`.

```
LValue isProvenValue(SpeculatedType provenType, SpeculatedType wantedType)
{
    if (!(provenType & ~wantedType))
        return m_out.booleanTrue;
    if (!(provenType & wantedType))
        return m_out.booleanFalse;
    return nullptr;
}
```

`provenType` and `wantedType` is **always different**. It always return false. Back to the speculateRealNumber function, in typeCheck, fail condition is **always false**.

double equal operation for NaN

It can be skipped. This is additional knowledge. How can doubleEqual filter out NaN?
/Source/JavaScriptCore/runtime/JSCJSVaueInlines.h # L1189

```
ALWAYS_INLINE bool JSValue::equalSlowCaseInline(JSGlobalObject* globalObject, JSValue v1, JSValue v2)
{
    VM& vm = getVM(globalObject);
```

```
    auto scope = DECLARE_THROW_SCOPE(vm);
    do {
        if (v1.isNumber()) {
            if (v2.isNumber())
                return v1.asNumber() == v2.asNumber();
            // Guaranteeing that if we have a number it is v2 makes some of the cases below simpler.
            std::swap(v1, v2);
        }
```

This code compiles like this:

```
callq       JSC::JSValue::asNumber
movsd       %xmm0, -0x158(%rbp)
leaq        -0x18(%rbp), %rdi
callq       JSC::JSValue::asNumber
movaps      %xmm0, %xmm1
movsd       -0x158(%rbp), %xmm0;
ucomisd     %xmm1, %xmm0            ; compare and set rflags (set PF if NaN)
sete        %al                     ; set byte if ZF
setnp       %cl                     ; set byte if not PF
andb        %cl, %al                ; result => ZF && !PF
```

The `ucomisd` instruction compares double values. It sets the PF flag if the value to be compared is NaN, and `setnp` checks it and applies it to the comparison result. (andb %cl, %al)

About ucomisd
Unordered Compare Scalar Double-Precision Floating-Point Values and Set EFLAGS
result : unordered, greater than, less than, or equal
OF, SF, AF set to 0.
ZF(Zero Flag), PF(Parity Flag), CF(Carry Flag) set according to the result.

ZF, PF, CF states:
1 1 1 for unordered
0 0 0 for greater than
0 0 1 for less than
1 0 0 for equal

setnp:
set byte if not parity

Deep dive into optimization phases

The target webkit is the version just before the patch commit.
https://github.com/WebKit/WebKit/tree/7fa74ea6672d4fedaf33e3f775cf5db739a8c38c

1. JavaScript code

```
function test() {
    function object_is_opt(value) {
        const tmp = {p0: value};

        if (Object.is(value, NaN))
            return 0;

        return tmp;
    }

    object_is_opt(NaN);
```

```
    for (let i = 0; i < 0x20000; i++)
        object_is_opt(1.1);

    return isNaN(object_is_opt(NaN));
}

resultIsNaN = test();
if (resultIsNaN)
    throw "FAILED";
```

The result is : `Exception: FAILED`

2. Bytecode

generated bytecode

Comments are javascript code corresponding to the generated bytecode and short description.

```
object_is_opt#AATTmW:[0x1040f4240->0x104099700, NoneFunctionCall, 80]: 20 instructions (0 16-bit
instructions, 0 32-bit instructions, 8 instructions with metadata); 196 bytes (116 metadata bytes); 2
parameter(s); 16 callee register(s); 6 variable(s); scope at loc4

bb#1
Predecessors: [ ]
[   0] enter
[   1] get_scope          dst:loc4
[   3] mov                dst:loc5, src:loc4
[   6] check_traps

// tmp = {p0: value};
[   7] mov                dst:loc6, src:<JSValue()>(const0)
[  10] new_object         dst:loc7, inlineCapacity:1
[  14] put_by_id          base:loc7, property:0, value:arg1, flags:IsDirect
[  20] mov                dst:loc6, src:loc7                        // loc6 = tmp

// Object.is(value, NaN);
[  23] resolve_scope      dst:loc10, scope:loc4, var:1, resolveType:GlobalProperty, localScopeDepth:0
[  30] get_from_scope     dst:loc11, scope:loc10, var:1,
getPutInfo:2048<ThrowIfNotFound|GlobalProperty|NotInitialization|NotStrictMode>, localScopeDepth:0,
offset:0
[  38] mov                dst:loc10, src:loc11
[  41] get_by_id          dst:loc7, base:loc10, property:2          // loc7 = Object.is
[  46] mov                dst:loc9, src:arg1                        // loc9 = value
[  49] resolve_scope      dst:loc8, scope:loc4, var:3, resolveType:GlobalProperty, localScopeDepth:0
[  56] get_from_scope     dst:loc11, scope:loc8, var:3,
getPutInfo:2048<ThrowIfNotFound|GlobalProperty|NotInitialization|NotStrictMode>, localScopeDepth:0,
offset:0
[  64] mov                dst:loc8, src:loc11                       // loc8 = NaN
[  67] call               dst:loc7, callee:loc7, argc:3, argv:16    // Object.is(value, NaN);

// Accoding to result of call instruction,
// go to bb#3 if true, go to bb#2 if false
[  73] jfalse             condition:loc7, targetLabel:5(->78)
Successors: [ #3 #2 ]

bb#2
Predecessors: [ #1 ]
// return 0;
[  76] ret                value:Int32: 0(const1)
Successors: [ ]

bb#3
Predecessors: [ #1 ]
// return tmp;
[  78] ret                value:loc6
Successors: [ ]
```

```
Identifiers:
  id0 = p0
  id1 = Object
  id2 = is
  id3 = NaN

Constants:
   k0 = <JSValue()>
   k1 = Int32: 0: in source as integer
```

How interpreter execute them

/Source/JavaScriptCore/runtime/ObjectConstructor.cpp # L883

```
JSC_DEFINE_HOST_FUNCTION(objectConstructorIs, (JSGlobalObject* globalObject, CallFrame* callFrame))
{
    return JSValue::encode(jsBoolean(sameValue(globalObject, callFrame->argument(0), callFrame-
>argument(1))));
}
```

This is the runtime code that handles the built-in function `Object.is`. It calls sameValue.

/Source/JavaScriptCore/runtime/JSCJSValueInlines.h # L1426

```
// See section 7.2.9: https://tc39.github.io/ecma262/#sec-samevalue
ALWAYS_INLINE bool sameValue(JSGlobalObject* globalObject, JSValue a, JSValue b)
{
    if (!a.isNumber())
        return JSValue::strictEqual(globalObject, a, b);
    if (!b.isNumber())
        return false;
    double x = a.asNumber();
    double y = b.asNumber();
    bool xIsNaN = std::isnan(x);
    bool yIsNaN = std::isnan(y);
    if (xIsNaN || yIsNaN)
        return xIsNaN && yIsNaN;
    return bitwise_cast<uint64_t>(x) == bitwise_cast<uint64_t>(y);
}
```

sameValue is implemented according to the ecma standard.

- https://tc39.es/ecma262/#sec-samevalue
- https://tc39.es/ecma262/#sec-numeric-types-number-sameValue

When the interpreter processes `sameValue(a,b)` for a Number type, it returns true if a and b are the same number or both are NaN.

Now let's see how this sameValue changes per compilation layer.

3. Dataflow graph

ByteCodeParser's work

ByteCodeParser compile the dataflow graph from a CodeBlock.
/Source/JavaScriptCore/dfg/DFGByteCodeParser.cpp # L2972

```
bool ByteCodeParser::handleIntrinsicCall(Node* callee, Operand result, Intrinsic intrinsic, int
registerOffset, int argumentCountIncludingThis, SpeculatedType prediction, const ChecksFunctor&
insertChecks)
        ...
        case ObjectIsIntrinsic: {
            if (argumentCountIncludingThis < 3)
                return false;
```

```
            insertChecks();
            setResult(addToGraph(SameValue, get(virtualRegisterForArgumentIncludingThis(1,
registerOffset)), get(virtualRegisterForArgumentIncludingThis(2, registerOffset))));
            return true;
        }
```

The Call instruction `Object.is` compiled to `SameValue` node.

compiled graph

```
        0: Beginning DFG phase live catch variable preservation phase.
        0: Before live catch variable preservation phase:

        0: DFG for object_is_opt#AATTmW:[0x1040f4360->0x1040f4240->0x104099700, DFGFunctionCall, 80]:
        0:   Fixpoint state: BeforeFixpoint; Form: LoadStore; Unification state: LocallyUnified; Ref
count state: EverythingIsLive
        0:   Arguments for block#0: D@0, D@1

     0  0: Block #0 (bc#0): (OSR target)
     0  0:   Execution count: 1.000000
     0  0:   Predecessors:
     0  0:   Successors: #1 #2
     0  0:   States: StructuresAreWatched, CurrentlyCFAUnreachable
     0  0:   Vars Before: <empty>
     0  0:   Intersected Vars Before: arg1:(FullTop, TOP, TOP, none:StructuresAreClobbered) arg0:
(FullTop, TOP, TOP, none:StructuresAreClobbered) loc0:(FullTop, TOP, TOP,
none:StructuresAreClobbered) loc1:(FullTop, TOP, TOP, none:StructuresAreClobbered) loc2:(FullTop,
TOP, TOP, none:StructuresAreClobbered) loc3:(FullTop, TOP, TOP, none:StructuresAreClobbered) loc4:
(FullTop, TOP, TOP, none:StructuresAreClobbered) loc5:(FullTop, TOP, TOP,
none:StructuresAreClobbered) loc6:(FullTop, TOP, TOP, none:StructuresAreClobbered) loc7:(FullTop,
TOP, TOP, none:StructuresAreClobbered) loc8:(FullTop, TOP, TOP, none:StructuresAreClobbered) loc9:
(FullTop, TOP, TOP, none:StructuresAreClobbered) loc10:(FullTop, TOP, TOP,
none:StructuresAreClobbered) loc11:(FullTop, TOP, TOP, none:StructuresAreClobbered) loc12:(FullTop,
TOP, TOP, none:StructuresAreClobbered) loc13:(FullTop, TOP, TOP, none:StructuresAreClobbered) loc14:
(FullTop, TOP, TOP, none:StructuresAreClobbered) loc15:(FullTop, TOP, TOP,
none:StructuresAreClobbered)
     0  0:   Var Links:
  0  0  0:    D@0:< 1:->        SetArgumentDefinitely(this(A~/FlushedJSValue), W:SideState, bc#0,
ExitValid)  predicting None
  1  0  0:    D@1:< 1:->        SetArgumentDefinitely(arg1(B~/FlushedJSValue), W:SideState, bc#0,
ExitValid)  predicting None
  2  0  0:    D@2:< 1:->        JSConstant(JS|PureInt, Undefined, bc#0, ExitValid)
  3  0  0:    D@3:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc0, W:SideState, ClobbersExit,
bc#0, ExitValid)
  4  0  0:    D@4:< 1:->        SetLocal(Check:Untyped:D@2, loc0(C~/FlushedJSValue), W:Stack(loc0),
bc#0, ExitInvalid)  predicting None
  5  0  0:    D@5:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc1, W:SideState, ClobbersExit,
bc#0, ExitInvalid)
  6  0  0:    D@6:< 1:->        SetLocal(Check:Untyped:D@2, loc1(D~/FlushedJSValue), W:Stack(loc1),
bc#0, ExitInvalid)  predicting None
  7  0  0:    D@7:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc2, W:SideState, ClobbersExit,
bc#0, ExitInvalid)
  8  0  0:    D@8:< 1:->        SetLocal(Check:Untyped:D@2, loc2(E~/FlushedJSValue), W:Stack(loc2),
bc#0, ExitInvalid)  predicting None
  9  0  0:    D@9:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc3, W:SideState, ClobbersExit,
bc#0, ExitInvalid)
 10  0  0:   D@10:< 1:->        SetLocal(Check:Untyped:D@2, loc3(F~/FlushedJSValue), W:Stack(loc3),
bc#0, ExitInvalid)  predicting None
 11  0  0:   D@11:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc4, W:SideState, ClobbersExit,
bc#0, ExitInvalid)
 12  0  0:   D@12:< 1:->        SetLocal(Check:Untyped:D@2, loc4(G~/FlushedJSValue), W:Stack(loc4),
bc#0, ExitInvalid)  predicting None
 13  0  0:   D@13:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc5, W:SideState, ClobbersExit,
bc#0, ExitInvalid)
 14  0  0:   D@14:< 1:->        SetLocal(Check:Untyped:D@2, loc5(H~/FlushedJSValue), W:Stack(loc5),
bc#0, ExitInvalid)  predicting None
 15  0  0:   D@15:< 1:->        JSConstant(JS|PureInt, Weak:Object: 0x10408dea0 with butterfly
```

0x0(base=0xffffffffffffffff8) (Structure %CG:Function), StructureID: 22096, bc#1, ExitValid)
 16 0 0:   D@16:< 1:->   JSConstant(JS|UseAsOther, Weak:Object: 0x103034428 with butterfly 0x0(base=0xffffffffffffffff8) (Structure %CD:JSGlobalLexicalEnvironment), StructureID: 21424, bc#1, ExitValid)
 17 0 0:   D@17:<!0:->   MovHint(Check:Untyped:D@16, MustGen, loc4, W:SideState, ClobbersExit, bc#1, ExitValid)
 18 0 0:   D@18:< 1:->   SetLocal(Check:Untyped:D@16, loc4(I~/FlushedJSValue), W:Stack(loc4), bc#1, exit: bc#3, ExitValid) predicting None
 19 0 0:   D@19:<!0:->   MovHint(Check:Untyped:D@16, MustGen, loc5, W:SideState, ClobbersExit, bc#3, ExitValid)
 20 0 0:   D@20:< 1:->   SetLocal(Check:Untyped:D@16, loc5(J~/FlushedJSValue), W:Stack(loc5), bc#3, exit: bc#6, ExitValid) predicting None
 21 0 0:   D@21:<!0:->   InvalidationPoint(MustGen, W:SideState, Exits, bc#6, ExitValid)
 22 0 0:   D@22:< 1:->   JSConstant(JS|PureInt, <JSValue()>, bc#7, ExitValid)
 23 0 0:   D@23:<!0:->   MovHint(Check:Untyped:D@22, MustGen, loc6, W:SideState, ClobbersExit, bc#7, ExitValid)
 24 0 0:   D@24:< 1:->   SetLocal(Check:Untyped:D@22, loc6(K~/FlushedJSValue), W:Stack(loc6), bc#7, exit: bc#10, ExitValid) predicting None
 25 0 0:   D@25:< 1:->   NewObject(JS|UseAsOther, %AA:Object, R:HeapObjectCount, W:HeapObjectCount, Exits, bc#10, ExitValid)
 26 0 0:   D@26:<!0:->   MovHint(Check:Untyped:D@25, MustGen, loc7, W:SideState, ClobbersExit, bc#10, ExitValid)
 27 0 0:   D@27:< 1:->   SetLocal(Check:Untyped:D@25, loc7(L~/FlushedJSValue), W:Stack(loc7), bc#10, exit: bc#14, ExitValid) predicting None
 28 0 0:   D@28:<!0:->   GetLocal(JS|MustGen|UseAsOther, arg1(B~/FlushedJSValue), R:Stack(arg1), bc#14, ExitValid) predicting None
 29 0 0:   D@29:<!0:->   FilterPutByStatus(Check:Untyped:D@25, MustGen, (<id='uid:(p0)', Transition: [0x300008bd0:[0x8bd0/35792, Object, (0/2, 0/0){}, NonArray, Proto:0x103011968]] to 0x300008c40:[0x8c40/35904, Object, (1/2, 0/0){p0:0}, NonArray, Proto:0x103011968, Leaf], [[]], offset = 0, >), W:SideState, bc#14, ExitValid)
 30 0 0:   D@30:<!0:->   CheckStructure(Check:Untyped:D@25, MustGen, [%AA:Object], R:JSCell_structureID, Exits, bc#14, ExitValid)
 31 0 0:   D@31:<!0:->   PutByOffset(Check:Untyped:D@25, Check:Untyped:D@25, Check:Untyped:D@28, MustGen, id0{p0}, 0, W:NamedProperties(0), ClobbersExit, bc#14, ExitValid)
 32 0 0:   D@32:<!0:->   PutStructure(Check:Untyped:D@25, MustGen, %AA:Object -> %A7:Object, ID:35904, R:JSObject_butterfly, W:JSCell_indexingType,JSCell_structureID,JSCell_typeInfoFlags, ClobbersExit, bc#14, ExitInvalid)
 33 0 0:   D@33:<!0:->   MovHint(Check:Untyped:D@25, MustGen, loc6, W:SideState, ClobbersExit, bc#20, ExitValid)
 34 0 0:   D@34:< 1:->   SetLocal(Check:Untyped:D@25, loc6(M~/FlushedJSValue), W:Stack(loc6), bc#20, exit: bc#23, ExitValid) predicting None
 35 0 0:   D@35:< 1:->   JSConstant(JS|UseAsOther, Weak:Object: 0x1040c0068 with butterfly 0x8004014408(base=0x8004014000) (Structure %Aq:global), StructureID: 33440, bc#23, ExitValid)
 36 0 0:   D@36:<!0:->   MovHint(Check:Untyped:D@35, MustGen, loc10, W:SideState, ClobbersExit, bc#23, ExitValid)
 37 0 0:   D@37:<!0:->   Phantom(Check:Untyped:D@16, MustGen, bc#23, ExitInvalid)
 38 0 0:   D@38:< 1:->   SetLocal(Check:Untyped:D@35, loc10(N~/FlushedJSValue), W:Stack(loc10), bc#23, exit: bc#30, ExitValid) predicting None
 39 0 0:   D@39:< 1:->   JSConstant(JS|UseAsOther, Weak:Object: 0x1040c0068 with butterfly 0x8004014408(base=0x8004014000) (Structure %Aq:global), StructureID: 33440, bc#30, ExitValid)
 40 0 0:   D@40:<!0:->   Phantom(Check:Untyped:D@39, MustGen, bc#30, ExitValid)
 41 0 0:   D@41:< 1:->   JSConstant(JS|UseAsOther, Weak:Object: 0x103038c88 with butterfly 0x8004008448(base=0x80040083c0) (Structure %Bu:Function), StructureID: 36016, bc#30, ExitValid)
 42 0 0:   D@42:<!0:->   Phantom(Check:Untyped:D@35, MustGen, bc#30, ExitValid)
 43 0 0:   D@43:<!0:->   MovHint(Check:Untyped:D@41, MustGen, loc11, W:SideState, ClobbersExit, bc#30, ExitValid)
 44 0 0:   D@44:< 1:->   SetLocal(Check:Untyped:D@41, loc11(O~/FlushedJSValue), W:Stack(loc11), bc#30, exit: bc#38, ExitValid) predicting None
 45 0 0:   D@45:<!0:->   MovHint(Check:Untyped:D@41, MustGen, loc10, W:SideState, ClobbersExit, bc#38, ExitValid)
 46 0 0:   D@46:< 1:->   SetLocal(Check:Untyped:D@41, loc10(P~/FlushedJSValue), W:Stack(loc10), bc#38, exit: bc#41, ExitValid) predicting None
 47 0 0:   D@47:<!0:->   FilterGetByStatus(Check:Untyped:D@41, MustGen, (Simple, <id='uid:(is)', [0x300008cb0:[0x8cb0/36016, Function, (0/0, 14/16){length:64, name:65, prototype:66, getPrototypeOf:67, getOwnPropertyDescriptor:68, getOwnPropertyNames:69, getOwnPropertySymbols:70, keys:71, defineProperty:72, create:73, values:74, hasOwn:75, hasOwn:76, is:77}, NonArray, Proto:0x103038508, Leaf]], [], offset = 77>, seenInJIT = true), W:SideState, bc#41, ExitValid)

```
 48  0  0:   D@48:<!0:->        Phantom(Check:Untyped:D@41, MustGen, bc#41, ExitValid)
 49  0  0:   D@49:<!0:->        CheckStructure(Check:Untyped:D@41, MustGen, [%Bu:Function],
R:JSCell_structureID, Exits, bc#41, ExitValid)
 50  0  0:   D@50:< 1:->        JSConstant(JS|UseAsOther, Weak:Object: 0x10408dec0 with butterfly
0x8004001fe8(base=0x8004001fc0) (Structure %Eq:Function), StructureID: 22432, bc#41, ExitValid)
 51  0  0:   D@51:<!0:->        MovHint(Check:Untyped:D@50, MustGen, loc7, W:SideState, ClobbersExit,
bc#41, ExitValid)
 52  0  0:   D@52:< 1:->        SetLocal(Check:Untyped:D@50, loc7(Q~/FlushedJSValue), W:Stack(loc7),
bc#41, exit: bc#46, ExitValid)  predicting None
 53  0  0:   D@53:<!0:->        MovHint(Check:Untyped:D@28, MustGen, loc9, W:SideState, ClobbersExit,
bc#46, ExitValid)
 54  0  0:   D@54:< 1:->        SetLocal(Check:Untyped:D@28, loc9(R~/FlushedJSValue), W:Stack(loc9),
bc#46, exit: bc#49, ExitValid)  predicting None
 55  0  0:   D@55:< 1:->        JSConstant(JS|UseAsOther, Weak:Object: 0x1040c0068 with butterfly
0x8004014408(base=0x8004014000) (Structure %Aq:global), StructureID: 33440, bc#49, ExitValid)
 56  0  0:   D@56:<!0:->        MovHint(Check:Untyped:D@55, MustGen, loc8, W:SideState, ClobbersExit,
bc#49, ExitValid)
 57  0  0:   D@57:<!0:->        Phantom(Check:Untyped:D@16, MustGen, bc#49, ExitInvalid)
 58  0  0:   D@58:< 1:->        SetLocal(Check:Untyped:D@55, loc8(S~/FlushedJSValue), W:Stack(loc8),
bc#49, exit: bc#56, ExitValid)  predicting None
 59  0  0:   D@59:<!0:->        Phantom(Check:Untyped:D@55, MustGen, bc#56, ExitValid)
 60  0  0:   D@60:< 1:->        JSConstant(JS|UseAsOther, Double: 9221120237041090560, nan, bc#56,
ExitValid)
 61  0  0:   D@61:<!0:->        MovHint(Check:Untyped:D@60, MustGen, loc11, W:SideState,
ClobbersExit, bc#56, ExitValid)
 62  0  0:   D@62:< 1:->        SetLocal(Check:Untyped:D@60, loc11(T~/FlushedJSValue),
W:Stack(loc11), bc#56, exit: bc#64, ExitValid)  predicting None
 63  0  0:   D@63:<!0:->        MovHint(Check:Untyped:D@60, MustGen, loc8, W:SideState, ClobbersExit,
bc#64, ExitValid)
 64  0  0:   D@64:< 1:->        SetLocal(Check:Untyped:D@60, loc8(U~/FlushedJSValue), W:Stack(loc8),
bc#64, exit: bc#67, ExitValid)  predicting None
 65  0  0:   D@65:<!0:->        FilterCallLinkStatus(Check:Untyped:D@50, MustGen, Statically Proved,
(Function: Object: 0x10408dec0 with butterfly 0x8004001fe8(base=0x8004001fc0) (Structure 0x3000057a0:
[0x57a0/22432, Function, (0/0, 2/4){length:64, name:65}, NonArray, Proto:0x103038508, Leaf]),
StructureID: 22432; Executable: NativeExecutable:0x10b5aca1c/0x10b3bea14), W:SideState, bc#67,
ExitValid)
 66  0  0:   D@66:<!0:->        CheckIsConstant(Check:Untyped:D@50, MustGen, Exits, bc#67, ExitValid)
 67  0  0:   D@67:<!0:->        Phantom(Check:Untyped:D@41, MustGen, bc#67, ExitValid)
 68  0  0:   D@68:< 1:->        SameValue(Check:Untyped:D@28, Check:Untyped:D@60, Boolean|UseAsOther,
Exits, bc#67, ExitValid)
 69  0  0:   D@69:<!0:->        MovHint(Check:Untyped:D@68, MustGen, loc7, W:SideState, ClobbersExit,
bc#67, ExitValid)
 70  0  0:   D@70:<!0:->        Phantom(Check:Untyped:D@50, MustGen, bc#67, ExitInvalid)
 71  0  0:   D@71:<!0:->        Phantom(Check:Untyped:D@41, MustGen, bc#67, ExitInvalid)
 72  0  0:   D@72:<!0:->        Phantom(Check:Untyped:D@28, MustGen, bc#67, ExitInvalid)
 73  0  0:   D@73:<!0:->        Phantom(Check:Untyped:D@60, MustGen, bc#67, ExitInvalid)
 74  0  0:   D@74:< 1:->        SetLocal(Check:Untyped:D@68, loc7(V~/FlushedJSValue), W:Stack(loc7),
bc#67, exit: bc#73, ExitValid)  predicting None
 75  0  0:   D@75:<!0:->        Branch(Check:Untyped:D@68, MustGen, T:#1, F:#2, W:SideState, bc#73,
ExitValid)
     0  0:   States: InvalidBranchDirection, StructuresAreWatched
     0  0:   Vars After: <empty>
     0  0:   Var Links: arg1:D@28 arg0:D@0 loc0:D@4 loc1:D@6 loc2:D@8 loc3:D@10 loc4:D@18 loc5:D@20
loc6:D@34 loc7:D@74 loc8:D@64 loc9:D@54 loc10:D@46 loc11:D@62

     1  0: Block #1 (bc#76):
     1  0:   Execution count: 1.000000
     1  0:   Predecessors: #0
     1  0:   Successors:
     1  0:   States: StructuresAreWatched, CurrentlyCFAUnreachable
     1  0:   Vars Before: <empty>
     1  0:   Intersected Vars Before: arg1:(FullTop, TOP, TOP, none:StructuresAreClobbered) arg0:
(FullTop, TOP, TOP, none:StructuresAreClobbered) loc0:(FullTop, TOP, TOP,
none:StructuresAreClobbered) loc1:(FullTop, TOP, TOP, none:StructuresAreClobbered) loc2:(FullTop,
TOP, TOP, none:StructuresAreClobbered) loc3:(FullTop, TOP, TOP, none:StructuresAreClobbered) loc4:
(FullTop, TOP, TOP, none:StructuresAreClobbered) loc5:(FullTop, TOP, TOP,
none:StructuresAreClobbered) loc6:(FullTop, TOP, TOP, none:StructuresAreClobbered) loc7:(FullTop,
```

```
TOP, TOP, none:StructuresAreClobbered) loc8:(FullTop, TOP, TOP, none:StructuresAreClobbered) loc9:
(FullTop, TOP, TOP, none:StructuresAreClobbered) loc10:(FullTop, TOP, TOP,
none:StructuresAreClobbered) loc11:(FullTop, TOP, TOP, none:StructuresAreClobbered) loc12:(FullTop,
TOP, TOP, none:StructuresAreClobbered) loc13:(FullTop, TOP, TOP, none:StructuresAreClobbered) loc14:
(FullTop, TOP, TOP, none:StructuresAreClobbered) loc15:(FullTop, TOP, TOP,
none:StructuresAreClobbered)
     1  0:   Var Links:
  0  1  0:   D@76:< 1:->        JSConstant(JS|UseAsOther, Int32: 0, bc#76, ExitValid)
  1  1  0:   D@77:<!0:->        Return(Check:Untyped:D@76, MustGen, W:SideState, Exits, bc#76,
ExitValid)
  2  1  0:   D@78:<!0:->        Flush(MustGen, arg1(W~/FlushedJSValue), R:Stack(arg1), W:SideState,
bc#76, ExitValid)  predicting None
  3  1  0:   D@79:<!0:->        Flush(MustGen, this(X~/FlushedJSValue), R:Stack(this), W:SideState,
bc#76, ExitValid)  predicting None
     1  0:   States: InvalidBranchDirection, StructuresAreWatched
     1  0:   Vars After: <empty>
     1  0:   Var Links: arg1:D@78 arg0:D@79

     2  0: Block #2 (bc#78):
     2  0:   Execution count: 1.000000
     2  0:   Predecessors: #0
     2  0:   Successors:
     2  0:   States: StructuresAreWatched, CurrentlyCFAUnreachable
     2  0:   Vars Before: <empty>
     2  0:   Intersected Vars Before: arg1:(FullTop, TOP, TOP, none:StructuresAreClobbered) arg0:
(FullTop, TOP, TOP, none:StructuresAreClobbered) loc0:(FullTop, TOP, TOP,
none:StructuresAreClobbered) loc1:(FullTop, TOP, TOP, none:StructuresAreClobbered) loc2:(FullTop,
TOP, TOP, none:StructuresAreClobbered) loc3:(FullTop, TOP, TOP, none:StructuresAreClobbered) loc4:
(FullTop, TOP, TOP, none:StructuresAreClobbered) loc5:(FullTop, TOP, TOP,
none:StructuresAreClobbered) loc6:(FullTop, TOP, TOP, none:StructuresAreClobbered) loc7:(FullTop,
TOP, TOP, none:StructuresAreClobbered) loc8:(FullTop, TOP, TOP, none:StructuresAreClobbered) loc9:
(FullTop, TOP, TOP, none:StructuresAreClobbered) loc10:(FullTop, TOP, TOP,
none:StructuresAreClobbered) loc11:(FullTop, TOP, TOP, none:StructuresAreClobbered) loc12:(FullTop,
TOP, TOP, none:StructuresAreClobbered) loc13:(FullTop, TOP, TOP, none:StructuresAreClobbered) loc14:
(FullTop, TOP, TOP, none:StructuresAreClobbered) loc15:(FullTop, TOP, TOP,
none:StructuresAreClobbered)
     2  0:   Var Links:
  0  2  0:   D@80:<!0:->        GetLocal(JS|MustGen|UseAsOther, loc6(Y~/FlushedJSValue),
R:Stack(loc6), bc#78, ExitValid)  predicting None
  1  2  0:   D@81:<!0:->        Return(Check:Untyped:D@80, MustGen, W:SideState, Exits, bc#78,
ExitValid)
  2  2  0:   D@82:<!0:->        Flush(MustGen, arg1(Z~/FlushedJSValue), R:Stack(arg1), W:SideState,
bc#78, ExitValid)  predicting None
  3  2  0:   D@83:<!0:->        Flush(MustGen, this(AB~/FlushedJSValue), R:Stack(this), W:SideState,
bc#78, ExitValid)  predicting None
     2  0:   States: InvalidBranchDirection, StructuresAreWatched
     2  0:   Vars After: <empty>
     2  0:   Var Links: arg1:D@82 arg0:D@83 loc6:D@80
```

The key part is:

```
// arguments is this and arg1(=value).
// It is being predicted as None.
  0  0  0:    D@0:< 1:->        SetArgumentDefinitely(this(A~/FlushedJSValue), W:SideState, bc#0,
ExitValid)  predicting None
  1  0  0:    D@1:< 1:->        SetArgumentDefinitely(arg1(B~/FlushedJSValue), W:SideState, bc#0,
ExitValid)  predicting None
  2  0  0:    D@2:< 1:->        JSConstant(JS|PureInt, Undefined, bc#0, ExitValid)
  3  0  0:    D@3:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc0, W:SideState, ClobbersExit,
bc#0, ExitValid)
  4  0  0:    D@4:< 1:->        SetLocal(Check:Untyped:D@2, loc0(C~/FlushedJSValue), W:Stack(loc0),
bc#0, ExitInvalid)  predicting None
  5  0  0:    D@5:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc1, W:SideState, ClobbersExit,
bc#0, ExitInvalid)
  6  0  0:    D@6:< 1:->        SetLocal(Check:Untyped:D@2, loc1(D~/FlushedJSValue), W:Stack(loc1),
bc#0, ExitInvalid)  predicting None
  7  0  0:    D@7:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc2, W:SideState, ClobbersExit,
```

```
bc#0, ExitInvalid)
  8  0  0:    D@8:< 1:->        SetLocal(Check:Untyped:D@2, loc2(E~/FlushedJSValue), W:Stack(loc2),
bc#0, ExitInvalid)  predicting None
  9  0  0:    D@9:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc3, W:SideState, ClobbersExit,
bc#0, ExitInvalid)
 10  0  0:   D@10:< 1:->        SetLocal(Check:Untyped:D@2, loc3(F~/FlushedJSValue), W:Stack(loc3),
bc#0, ExitInvalid)  predicting None
 11  0  0:   D@11:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc4, W:SideState, ClobbersExit,
bc#0, ExitInvalid)
 12  0  0:   D@12:< 1:->        SetLocal(Check:Untyped:D@2, loc4(G~/FlushedJSValue), W:Stack(loc4),
bc#0, ExitInvalid)  predicting None
 13  0  0:   D@13:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc5, W:SideState, ClobbersExit,
bc#0, ExitInvalid)
 14  0  0:   D@14:< 1:->        SetLocal(Check:Untyped:D@2, loc5(H~/FlushedJSValue), W:Stack(loc5),
bc#0, ExitInvalid)  predicting None


// Create tmp object.
 25  0  0:   D@25:< 1:->        NewObject(JS|UseAsOther, %AA:Object, R:HeapObjectCount,
W:HeapObjectCount, Exits, bc#10, ExitValid)
 26  0  0:   D@26:<!0:->        MovHint(Check:Untyped:D@25, MustGen, loc7, W:SideState, ClobbersExit,
bc#10, ExitValid)
 27  0  0:   D@27:< 1:->        SetLocal(Check:Untyped:D@25, loc7(L~/FlushedJSValue), W:Stack(loc7),
bc#10, exit: bc#14, ExitValid)  predicting None


// Set value in tmp.p0
 28  0  0:   D@28:<!0:->        GetLocal(JS|MustGen|UseAsOther, arg1(B~/FlushedJSValue),
R:Stack(arg1), bc#14, ExitValid)  predicting None
 29  0  0:   D@29:<!0:->        FilterPutByStatus(Check:Untyped:D@25, MustGen, (<id='uid:(p0)',
Transition: [0x300008bd0:[0x8bd0/35792, Object, (0/2, 0/0){}, NonArray, Proto:0x103011968]] to
0x300008c40:[0x8c40/35904, Object, (1/2, 0/0){p0:0}, NonArray, Proto:0x103011968, Leaf], [[]], offset
= 0, >), W:SideState, bc#14, ExitValid)
 30  0  0:   D@30:<!0:->        CheckStructure(Check:Untyped:D@25, MustGen, [%AA:Object],
R:JSCell_structureID, Exits, bc#14, ExitValid)
 31  0  0:   D@31:<!0:->        PutByOffset(Check:Untyped:D@25, Check:Untyped:D@25,
Check:Untyped:D@28, MustGen, id0{p0}, 0, W:NamedProperties(0), ClobbersExit, bc#14, ExitValid)
 32  0  0:   D@32:<!0:->        PutStructure(Check:Untyped:D@25, MustGen, %AA:Object -> %A7:Object,
ID:35904, R:JSObject_butterfly, W:JSCell_indexingType,JSCell_structureID,JSCell_typeInfoFlags,
ClobbersExit, bc#14, ExitInvalid)


// NaN
 58  0  0:   D@58:< 1:->        SetLocal(Check:Untyped:D@55, loc8(S~/FlushedJSValue), W:Stack(loc8),
bc#49, exit: bc#56, ExitValid)  predicting None
 59  0  0:   D@59:<!0:->        Phantom(Check:Untyped:D@55, MustGen, bc#56, ExitValid)
 60  0  0:   D@60:< 1:->        JSConstant(JS|UseAsOther, Double: 9221120237041090560, nan, bc#56,
ExitValid)
 61  0  0:   D@61:<!0:->        MovHint(Check:Untyped:D@60, MustGen, loc11, W:SideState,
ClobbersExit, bc#56, ExitValid)
 62  0  0:   D@62:< 1:->        SetLocal(Check:Untyped:D@60, loc11(T~/FlushedJSValue),
W:Stack(loc11), bc#56, exit: bc#64, ExitValid)  predicting None


// D@50 is Object.is function.
// Check that Object.is has not changed.
// Whether to use internal implementation code that has not been overitten by the user
 64  0  0:   D@64:< 1:->        SetLocal(Check:Untyped:D@60, loc8(U~/FlushedJSValue), W:Stack(loc8),
bc#64, exit: bc#67, ExitValid)  predicting None
 65  0  0:   D@65:<!0:->        FilterCallLinkStatus(Check:Untyped:D@50, MustGen, Statically Proved,
(Function: Object: 0x10408dec0 with butterfly 0x8004001fe8(base=0x8004001fc0) (Structure 0x3000057a0:
[0x57a0/22432, Function, (0/0, 2/4){length:64, name:65}, NonArray, Proto:0x103038508, Leaf]),
StructureID: 22432; Executable: NativeExecutable:0x10b5aca1c/0x10b3bea14), W:SideState, bc#67,
ExitValid)
 66  0  0:   D@66:<!0:->        CheckIsConstant(Check:Untyped:D@50, MustGen, Exits, bc#67, ExitValid)
 67  0  0:   D@67:<!0:->        Phantom(Check:Untyped:D@41, MustGen, bc#67, ExitValid)
// Builtin JS function can be inlined.
```

```
// Object.is is inlined with SameValue.
 68  0  0:   D@68:< 1:->        SameValue(Check:Untyped:D@28, Check:Untyped:D@60, Boolean|UseAsOther,
Exits, bc#67, ExitValid)
 69  0  0:   D@69:<!0:->        MovHint(Check:Untyped:D@68, MustGen, loc7, W:SideState, ClobbersExit,
bc#67, ExitValid)
 70  0  0:   D@70:<!0:->        Phantom(Check:Untyped:D@50, MustGen, bc#67, ExitInvalid)
 71  0  0:   D@71:<!0:->        Phantom(Check:Untyped:D@41, MustGen, bc#67, ExitInvalid)
 72  0  0:   D@72:<!0:->        Phantom(Check:Untyped:D@28, MustGen, bc#67, ExitInvalid)
 73  0  0:   D@73:<!0:->        Phantom(Check:Untyped:D@60, MustGen, bc#67, ExitInvalid)


// Depending on the result of SameValue,
// it branches to #1 block if true and to #2 block if false.
 74  0  0:   D@74:< 1:->        SetLocal(Check:Untyped:D@68, loc7(V~/FlushedJSValue), W:Stack(loc7),
bc#67, exit: bc#73, ExitValid)  predicting None
 75  0  0:   D@75:<!0:->        Branch(Check:Untyped:D@68, MustGen, T:#1, F:#2, W:SideState, bc#73,
ExitValid)


    1  0: Block #1 (bc#78):
// return 0;
  0  1  0:   D@76:< 1:->        JSConstant(JS|UseAsOther, Int32: 0, bc#76, ExitValid)
  1  1  0:   D@77:<!0:->        Return(Check:Untyped:D@76, MustGen, W:SideState, Exits, bc#76,
ExitValid)
  2  1  0:   D@78:<!0:->        Flush(MustGen, arg1(W~/FlushedJSValue), R:Stack(arg1), W:SideState,
bc#76, ExitValid)  predicting None
  3  1  0:   D@79:<!0:->        Flush(MustGen, this(X~/FlushedJSValue), R:Stack(this), W:SideState,
bc#76, ExitValid)  predicting None


    2  0: Block #2 (bc#80):
// return tmp;
  0  2  0:   D@80:<!0:->        GetLocal(JS|MustGen|UseAsOther, loc6(Y~/FlushedJSValue),
R:Stack(loc6), bc#78, ExitValid)  predicting None
  1  2  0:   D@81:<!0:->        Return(Check:Untyped:D@80, MustGen, W:SideState, Exits, bc#78,
ExitValid)
  2  2  0:   D@82:<!0:->        Flush(MustGen, arg1(Z~/FlushedJSValue), R:Stack(arg1), W:SideState,
bc#78, ExitValid)  predicting None
  3  2  0:   D@83:<!0:->        Flush(MustGen, this(AB~/FlushedJSValue), R:Stack(this), W:SideState,
bc#78, ExitValid)  predicting None
```

... more key part is:

```
// arg1(=value)
 28  0  0:   D@28:<!0:->        GetLocal(JS|MustGen|UseAsOther, arg1(B~/FlushedJSValue),
R:Stack(arg1), bc#14, ExitValid)  predicting None

// NaN
 60  0  0:   D@60:< 1:->        JSConstant(JS|UseAsOther, Double: 9221120237041090560, nan, bc#56,
ExitValid)

// Object.is(value, NaN) ?
 68  0  0:   D@68:< 1:->        SameValue(Check:Untyped:D@28, Check:Untyped:D@60, Boolean|UseAsOther,
Exits, bc#67, ExitValid)

 75  0  0:   D@75:<!0:->        Branch(Check:Untyped:D@68, MustGen, T:#1, F:#2, W:SideState, bc#73,
ExitValid)

// true
  0  1  0:   D@76:< 1:->        JSConstant(JS|UseAsOther, Int32: 0, bc#76, ExitValid)
  1  1  0:   D@77:<!0:->        Return(Check:Untyped:D@76, MustGen, W:SideState, Exits, bc#76,
ExitValid)

// false
  0  2  0:   D@80:<!0:->        GetLocal(JS|MustGen|UseAsOther, loc6(Y~/FlushedJSValue),
R:Stack(loc6), bc#78, ExitValid)  predicting None
```

```
  1  2  0:   D@81:<!0:->        Return(Check:Untyped:D@80, MustGen, W:SideState, Exits, bc#78,
ExitValid)
```

Let's analyze the optimization phases, focusing on how this code changes.

4. Fixup

> Fix portions of the graph that are inefficient given the predictions that we have. This should run after prediction propagation but before CSE.

This phase inserts type conversions if needed.

Before and After

```
// tmp = {p0: value}
  25  0  X:   D@25:< 1:->       NewObject(JS|UseAsOther, Final, %Db:Object, R:HeapObjectCount,
W:HeapObjectCount, Exits, bc#10, ExitValid)
  26  0  X:   D@26:<!0:->       MovHint(Check:Untyped:D@25, MustGen, loc7, W:SideState, ClobbersExit,
bc#10, ExitValid)
  27  0  X:   D@27:< 1:->       SetLocal(Check:Untyped:D@25, loc7(L~<Final>/FlushedJSValue),
W:Stack(loc7), bc#10, exit: bc#14, ExitValid)  predicting Final


// D@28 is value.
// What was predicted as None is now predicted as NonIntAsDouble.
  28  0  X:   D@28:<!0:->       GetLocal(Check:Untyped:D@1, JS|MustGen|UseAsOther, NonIntAsDouble,
arg1(B~<Double>/FlushedJSValue), R:Stack(arg1), bc#14, ExitValid)  predicting NonIntAsDouble
  29  0  X:   D@29:<!0:->       FilterPutByStatus(Check:Untyped:D@25, MustGen, (<id='uid:(p0)',
Transition: [0x300008bd0:[0x8bd0/35792, Object, (0/2, 0/0){}, NonArray, Proto:0x104011968]] to
0x300008c40:[0x8c40/35904, Object, (1/2, 0/0){p0:0}, NonArray, Proto:0x104011968, Leaf], [[]], offset
= 0, >), W:SideState, bc#14, ExitValid)


// Known some use kind.
- 30  0  6:   D@30:<!0:->       CheckStructure(Check:Untyped:D@25, MustGen, [%Db:Object],
R:JSCell_structureID, Exits, bc#14, ExitValid)
- 31  0  6:   D@31:<!0:->       PutByOffset(Check:Untyped:D@25, Check:Untyped:D@25,
Check:Untyped:D@28, MustGen, id0{p0}, 0, W:NamedProperties(0), ClobbersExit, bc#14, ExitValid)
- 32  0  6:   D@32:<!0:->       PutStructure(Check:Untyped:D@25, MustGen, %Db:Object -> %Cd:Object,
ID:35904, R:JSObject_butterfly, W:JSCell_indexingType,JSCell_structureID,JSCell_typeInfoFlags,
ClobbersExit, bc#14, ExitInvalid)
+ 30  0  7:   D@30:<!0:->       CheckStructure(Check:Cell:D@25, MustGen, [%Db:Object],
R:JSCell_structureID, Exits, bc#14, ExitValid)
+ 31  0  7:   D@89:<!0:->       Check(Check:Number:D@28, MustGen, Exits, bc#14, ExitValid)
+ 32  0  7:   D@31:<!0:->       PutByOffset(Check:KnownCell:D@25, Check:KnownCell:D@25,
Check:Untyped:D@28, MustGen, id0{p0}, 0, W:NamedProperties(0), ClobbersExit, bc#14, ExitValid)
+ 33  0  7:   D@32:<!0:->       PutStructure(Check:KnownCell:D@25, MustGen, %Db:Object -> %Cd:Object,
ID:35904, R:JSObject_butterfly, W:JSCell_indexingType,JSCell_structureID,JSCell_typeInfoFlags,
ClobbersExit, bc#14, ExitInvalid)


// D@60 is NaN.
// It is being predicted as DoublePureNaN.
- 59  0  6:   D@59:<!0:->       Phantom(Check:Untyped:D@55, MustGen, bc#56, ExitValid)
+ 60  0  7:   D@59:<!0:->       Check(MustGen, bc#56, ExitValid)
  6X  0  X:   D@60:< 1:->       JSConstant(JS|UseAsOther, DoublePureNaN, Double: 9221120237041090560,
nan, bc#56, ExitValid)
  6X  0  X:   D@61:<!0:->       MovHint(Check:Untyped:D@60, MustGen, loc11, W:SideState,
ClobbersExit, bc#56, ExitValid)
  6X  0  X:   D@62:< 1:->       SetLocal(Check:Untyped:D@60, loc11(T~<Double>/FlushedJSValue),
W:Stack(loc11), bc#56, exit: bc#64, ExitValid)  predicting DoublePureNaN


// Check that Object.is is builtin JS function.
  6X  0  X:   D@65:<!0:->       FilterCallLinkStatus(Check:Untyped:D@50, MustGen, Statically Proved,
(Function: Object: 0x10608dec0 with butterfly 0x7001001fe8(base=0x7001001fc0) (Structure 0x3000057a0:
[0x57a0/22432, Function, (0/0, 2/4){length:64, name:65}, NonArray, Proto:0x104038508, Leaf]),
```

```
StructureID: 22432; Executable: NativeExecutable:0x10b5aca1c/0x10b3bea14), W:SideState, bc#67,
ExitValid)
// Known some use kind.
- 66  0  6:   D@66:<!0:->       CheckIsConstant(Check:Untyped:D@50, MustGen, Exits, bc#67, ExitValid)
- 67  0  6:   D@67:<!0:->       Phantom(Check:Untyped:D@41, MustGen, bc#67, ExitValid)
+ 67  0  7:   D@66:<!0:->       CheckIsConstant(Check:Cell:D@50, MustGen, <0x10608dec0, Function>,
<host function>, Exits, bc#67, ExitValid)
+ 68  0  7:   D@67:<!0:->       Check(MustGen, bc#67, ExitValid)


// Before fixup,
// D@28 is value and D@60 is NaN. There are Untyped.
// After fixup, value is converted to DoubleRep by profiling information.
// And NaN is converted to DoubleConstant.
// SameValue now knows the use kinds of children.
- 68  0  6:   D@68:< 1:->       SameValue(Check:Untyped:D@28, Check:Untyped:D@60, Boolean|UseAsOther,
Bool, Exits, bc#67, ExitValid)
+ 69  0  7:   D@90:< 1:->       DoubleRep(Check:RealNumber:D@28, Double|PureInt, BytecodeDouble,
Exits, bc#67, ExitValid)
+ 70  0  7:   D@91:< 1:->       DoubleConstant(Double|PureInt, BytecodeDouble, Double:
9221120237041090560, nan, bc#67, ExitValid)
+ 71  0  7:   D@68:< 1:->       SameValue(Check:DoubleRep:D@90<Double>, Check:DoubleRep:D@91<Double>,
Boolean|UseAsOther, Bool, Exits, bc#67, ExitValid)
  XX  0  X:   D@69:<!0:->       MovHint(Check:Untyped:D@68, MustGen, loc7, W:SideState, ClobbersExit,
bc#67, ExitValid)
- 70  0  6:   D@70:<!0:->       Phantom(Check:Untyped:D@50, MustGen, bc#67, ExitInvalid)
- 71  0  6:   D@71:<!0:->       Phantom(Check:Untyped:D@41, MustGen, bc#67, ExitInvalid)
- 72  0  6:   D@72:<!0:->       Phantom(Check:Untyped:D@28, MustGen, bc#67, ExitInvalid)
- 73  0  6:   D@73:<!0:->       Phantom(Check:Untyped:D@60, MustGen, bc#67, ExitInvalid)
+ 73  0  7:   D@70:<!0:->       Check(MustGen, bc#67, ExitInvalid)
+ 74  0  7:   D@71:<!0:->       Check(MustGen, bc#67, ExitInvalid)
+ 75  0  7:   D@72:<!0:->       Check(MustGen, bc#67, ExitInvalid)
+ 76  0  7:   D@73:<!0:->       Check(MustGen, bc#67, ExitInvalid)
  7X  0  X:   D@74:< 1:->       SetLocal(Check:Untyped:D@68, loc7(V~<Boolean>/FlushedJSValue),
W:Stack(loc7), bc#67, exit: bc#73, ExitValid)  predicting Bool


// #1 for true, #2 for false.
// Known some use kind.
- 75  0  6:   D@75:<!0:->       Branch(Check:Untyped:D@68, MustGen, T:#1/w:1.000000, F:#2/w:1.000000,
W:SideState, bc#73, ExitValid)
+ 78  0  7:   D@75:<!0:->       Branch(Check:KnownBoolean:D@68, MustGen, T:#1/w:1.000000,
F:#2/w:1.000000, W:SideState, bc#73, ExitValid)


      1  X: Block #1 (bc#76):
// return 0
   0  1  X:   D@76:< 1:->       JSConstant(JS|UseAsOther, BoolInt32, Int32: 0, bc#76, ExitValid)
   1  1  X:   D@77:<!0:->       Return(Check:Untyped:D@76, MustGen, W:SideState, Exits, bc#76,
ExitValid)
   2  1  X:   D@78:<!0:->       Flush(Check:Untyped:D@87, MustGen|IsFlushed, arg1(B~
<Double>/FlushedJSValue), R:Stack(arg1), W:SideState, bc#76, ExitValid)  predicting NonIntAsDouble
   3  1  X:   D@79:<!0:->       Flush(Check:Untyped:D@88, MustGen|IsFlushed, this(A~
<Other>/FlushedJSValue), R:Stack(this), W:SideState, bc#76, ExitValid)  predicting Other

      2  X: Block #2 (bc#78):
// return tmp
   0  2  X:   D@80:<!0:->       GetLocal(Check:Untyped:D@84, JS|MustGen|UseAsOther, Final, loc6(M~
<Final>/FlushedJSValue), R:Stack(loc6), bc#78, ExitValid)  predicting Final
   1  2  X:   D@81:<!0:->       Return(Check:Untyped:D@80, MustGen, W:SideState, Exits, bc#78,
ExitValid)
   2  2  X:   D@82:<!0:->       Flush(Check:Untyped:D@85, MustGen|IsFlushed, arg1(B~
<Double>/FlushedJSValue), R:Stack(arg1), W:SideState, bc#78, ExitValid)  predicting NonIntAsDouble
   3  2  X:   D@83:<!0:->       Flush(Check:Untyped:D@86, MustGen|IsFlushed, this(A~
<Other>/FlushedJSValue), R:Stack(this), W:SideState, bc#78, ExitValid)  predicting Other
```

See D@90 and D@91, by profiling information, the **value** is predicted as `NonIntAsDouble`, and **NaN** is a `JSConstant` with a double type value. Therefore, each opcode has been replaced with D@90 `DoubleRep` and D@91 `DoubleConstant` for a more efficient graph. In this case, value must be RealNumber. Of course, it is checked whether the value is of type RealNumber.

Now, the type of the children of D@68 SameValue is Double, and the return type is Boolean, which is also known to D@75 Branch.

Related code

/Source/JavaScriptCore/dfg/DFGFixupPhase.cpp # L4620

```
            // Now, insert type conversions if necessary.
            m_graph.doToChildren(
                node,
                [&] (Edge& edge) {
                    Node* result = nullptr;

                    switch (edge.useKind()) {
                    case DoubleRepUse:
                    case DoubleRepRealUse:
                    case DoubleRepAnyIntUse: {
                        if (edge->hasDoubleResult())
                            break;

                        ASSERT(indexForChecks != UINT_MAX);
                        if (edge->isNumberConstant()) {
                            result = m_insertionSet.insertNode(     // <--- insert our DoubleConstant
node.
                                indexForChecks, SpecBytecodeDouble, DoubleConstant, originForChecks,
                                OpInfo(m_graph.freeze(jsDoubleNumber(edge->asNumber()))));
                        } else if (edge->hasInt52Result()) {
                            result = m_insertionSet.insertNode(
                                indexForChecks, SpecAnyIntAsDouble, DoubleRep, originForChecks,
                                Edge(edge.node(), Int52RepUse));
                        } else {
                            UseKind useKind;
                            if (edge->shouldSpeculateDoubleReal())
                                useKind = RealNumberUse;
                            else if (edge->shouldSpeculateNumber())
                                useKind = NumberUse;
                            else
                                useKind = NotCellNorBigIntUse;

                            result = m_insertionSet.insertNode(     // <--- insert our DoubleRep
node.
                                indexForChecks, SpecBytecodeDouble, DoubleRep, originForChecks,
                                Edge(edge.node(), useKind));
                        }

                        edge.setNode(result);
                        break;
                    }
```

Type conversion for Children of SameValue is done.

5. Constant folding

> CFA-based constant folding. Walks those blocks marked by the CFA as having inferred constants, and replaces those nodes with constants whilst injecting Phantom nodes to keep the children alive (which is necessary for OSR exit).

Before and After

```
// D@60 is NaN, predicting DoublePureNaN
  60  0 1X:   D@59:<!0:->       Check(MustGen, bc#56, ExitValid)
  61  0 1X:   D@60:< 1:->       JSConstant(JS|UseAsOther, DoublePureNaN, Double: 9221120237041090560,
nan, bc#56, ExitValid)
  62  0 1X:   D@61:<!0:->       MovHint(Check:Untyped:D@60, MustGen, loc11, W:SideState,
ClobbersExit, bc#56, ExitValid)
  63  0 1X:   D@62:< 1:->       SetLocal(Check:Untyped:D@60, loc11(T~<Double>/FlushedJSValue),
W:Stack(loc11), bc#56, exit: bc#64, ExitValid)  predicting DoublePureNaN


// D@50 is Object.is
  66  0 1X:   D@65:<!0:->       FilterCallLinkStatus(Check:Untyped:D@50, MustGen, Statically Proved,
(Function: Object: 0x10608dec0 with butterfly 0x7001001fe8(base=0x7001001fc0) (Structure 0x3000057a0:
[0x57a0/22432, Function, (0/0, 2/4){length:64, name:65}, NonArray, Proto:0x104038508, Leaf]),
StructureID: 22432; Executable: NativeExecutable:0x10b5aca1c/0x10b3bea14), W:SideState, bc#67,
ExitValid)
- 67  0 12:   D@66:<!0:->       CheckIsConstant(Cell:D@50, MustGen, <0x10608dec0, Function>, <host
function>, Exits, bc#67, ExitValid)
+ 67  0 13:   D@66:<!0:->       Check(MustGen, bc#67, ExitValid)
  68  0 1X:   D@67:<!0:->       Check(MustGen, bc#67, ExitValid)


// Constant folding converts SameValue to JSConstant,
// because if D@28 is a RealNumber it will always have false.
  69  0 1X:   D@90:< 1:->       DoubleRep(Check:RealNumber:D@28, Double|PureInt, BytecodeDouble,
Exits, bc#67, ExitValid)
  70  0 1X:   D@91:< 1:->       DoubleConstant(Double|PureInt, BytecodeDouble, Double:
9221120237041090560, nan, bc#67, ExitValid)
- 71  0 12:   D@68:< 1:->       SameValue(DoubleRep:D@90<Double>, DoubleRep:D@91<Double>,
Boolean|UseAsOther, Bool, Exits, bc#67, ExitValid)
+ 71  0 13:   D@68:< 1:->       JSConstant(Boolean|UseAsOther, Bool, False, bc#67, ExitValid)
  72  0 1X:   D@69:<!0:->       MovHint(Check:Untyped:D@68, MustGen, loc7, W:SideState, ClobbersExit,
bc#67, ExitValid)
  73  0 1X:   D@70:<!0:->       Check(MustGen, bc#67, ExitInvalid)
  74  0 1X:   D@71:<!0:->       Check(MustGen, bc#67, ExitInvalid)
  75  0 1X:   D@72:<!0:->       Check(MustGen, bc#67, ExitInvalid)
  76  0 1X:   D@73:<!0:->       Check(MustGen, bc#67, ExitInvalid)
  77  0 1X:   D@74:< 1:->       SetLocal(Check:Untyped:D@68, loc7(V~<Boolean>/FlushedJSValue),
W:Stack(loc7), bc#67, exit: bc#73, ExitValid)  predicting Bool


  78  0 1X:   D@75:<!0:->       Branch(KnownBoolean:D@68, MustGen, T:#1/w:1.000000, F:#2/w:1.000000,
W:SideState, bc#73, ExitValid)


// true
      1 1X: Block #1 (bc#76):

   0  1 1X:   D@76:< 1:->       JSConstant(JS|UseAsOther, BoolInt32, Int32: 0, bc#76, ExitValid)
   1  1 1X:   D@77:<!0:->       Return(Check:Untyped:D@76, MustGen, W:SideState, Exits, bc#76,
ExitValid)
   2  1 1X:   D@78:<!0:->       Flush(Check:Untyped:D@87, MustGen|IsFlushed, arg1(B~
<Double>/FlushedJSValue), R:Stack(arg1), W:SideState, bc#76, ExitValid)  predicting NonIntAsDouble
   3  1 1X:   D@79:<!0:->       Flush(Check:Untyped:D@88, MustGen|IsFlushed, this(A~
<Other>/FlushedJSValue), R:Stack(this), W:SideState, bc#76, ExitValid)  predicting Other


// false
      2 1X: Block #2 (bc#78):

   0  2 1X:   D@80:<!0:->       GetLocal(Check:Untyped:D@84, JS|MustGen|UseAsOther, Final, loc6(M~
<Final>/FlushedJSValue), R:Stack(loc6), bc#78, ExitValid)  predicting Final
   1  2 1X:   D@81:<!0:->       Return(Check:Untyped:D@80, MustGen, W:SideState, Exits, bc#78,
ExitValid)
   2  2 1X:   D@82:<!0:->       Flush(Check:Untyped:D@85, MustGen|IsFlushed, arg1(B~
<Double>/FlushedJSValue), R:Stack(arg1), W:SideState, bc#78, ExitValid)  predicting NonIntAsDouble
```

```
   3  2 1X:   D@83:<!0:->       Flush(Check:Untyped:D@86, MustGen|IsFlushed, this(A~
<Other>/FlushedJSValue), R:Stack(this), W:SideState, bc#78, ExitValid)  predicting Other
```

Related code

[/Source/JavaScriptCore/dfg/DFGConstantFoldingPhase.cpp # L1127]

```
            // Interesting fact: this freezing that we do right here may turn an fragile value into
            // a weak value. See DFGValueStrength.h.
            FrozenValue* value = m_graph.freeze(m_state.forNode(node).value());
            if (!*value)
                continue;

            if (node->op() == GetLocal) {
                // Need to preserve bytecode liveness in ThreadedCPS form. This wouldn't be necessary
                // if it wasn't for https://bugs.webkit.org/show_bug.cgi?id=144086.
                m_insertionSet.insertNode(
                    indexInBlock, SpecNone, PhantomLocal, node->origin,
                    OpInfo(node->variableAccessData()));
                m_graph.dethread();
            } else
                m_insertionSet.insertCheck(m_graph, indexInBlock, node);
            m_graph.convertToConstant(node, value); // <--- SameValue to JSConstant.
```

6. CFG Simplification

> jump to single predecessor -> merge blocks
> branch on constant -> jump
> branch to same blocks -> jump
> jump-only block -> remove
> kill dead code

Before and After

```
// Branch on constant
- 78  0 13:   D@75:<!0:->       Branch(KnownBoolean:D@68, MustGen, T:#1/w:1.000000, F:#2/w:1.000000,
W:SideState, bc#73, ExitValid)
// Branch on constant has a known direction.
// Unused block is removed and used block is merged with the current block.
// In this case, always false block is taken, it is merged.
+ 78  0 14:   D@75:<!0:->       Check(MustGen, bc#73, ExitValid)
+ 79  0 14:   D@92:<!0:->       Flush(MustGen, this(A~<Other>/FlushedJSValue), R:Stack(this),
W:SideState, bc#73, ExitValid)  predicting Other
+ 80  0 14:   D@93:<!0:->       Flush(MustGen, arg1(B~<Double>/FlushedJSValue), R:Stack(arg1),
W:SideState, bc#73, ExitValid)  predicting NonIntAsDouble
+ 81  0 14:   D@80:<!0:->       GetLocal(Check:Untyped:D@84, JS|MustGen|UseAsOther, Final, loc6(M~
<Final>/FlushedJSValue), R:Stack(loc6), bc#78, ExitValid)  predicting Final
+ 82  0 14:   D@81:<!0:->       Return(Check:Untyped:D@80, MustGen, W:SideState, Exits, bc#78,
ExitValid)
+ 83  0 14:   D@82:<!0:->       Flush(Check:Untyped:D@85, MustGen|IsFlushed, arg1(B~
<Double>/FlushedJSValue), R:Stack(arg1), W:SideState, bc#78, ExitValid)  predicting NonIntAsDouble
+ 84  0 14:   D@83:<!0:->       Flush(Check:Untyped:D@86, MustGen|IsFlushed, this(A~
<Other>/FlushedJSValue), R:Stack(this), W:SideState, bc#78, ExitValid)  predicting Other


-     1 13: Block #1 (bc#76):

-  0  1 13:   D@76:< 1:->       JSConstant(JS|UseAsOther, BoolInt32, Int32: 0, bc#76, ExitValid)
-  1  1 13:   D@77:<!0:->       Return(Check:Untyped:D@76, MustGen, W:SideState, Exits, bc#76,
ExitValid)
-  2  1 13:   D@78:<!0:->       Flush(Check:Untyped:D@87, MustGen|IsFlushed, arg1(B~
<Double>/FlushedJSValue), R:Stack(arg1), W:SideState, bc#76, ExitValid)  predicting NonIntAsDouble
-  3  1 13:   D@79:<!0:->       Flush(Check:Untyped:D@88, MustGen|IsFlushed, this(A~
<Other>/FlushedJSValue), R:Stack(this), W:SideState, bc#76, ExitValid)  predicting Other


-     2 13: Block #2 (bc#78):
```

```
-  0  2 13:   D@80:<!0:->       GetLocal(Check:Untyped:D@84, JS|MustGen|UseAsOther, Final, loc6(M~
<Final>/FlushedJSValue), R:Stack(loc6), bc#78, ExitValid)  predicting Final
-  1  2 13:   D@81:<!0:->       Return(Check:Untyped:D@80, MustGen, W:SideState, Exits, bc#78,
ExitValid)
-  2  2 13:   D@82:<!0:->       Flush(Check:Untyped:D@85, MustGen|IsFlushed, arg1(B~
<Double>/FlushedJSValue), R:Stack(arg1), W:SideState, bc#78, ExitValid)  predicting NonIntAsDouble
-  3  2 13:   D@83:<!0:->       Flush(Check:Untyped:D@86, MustGen|IsFlushed, this(A~
<Other>/FlushedJSValue), R:Stack(this), W:SideState, bc#78, ExitValid)  predicting Other
```

Related code

/Source/JavaScriptCore/dfg/DFGCFGSimplificationPhase.cpp # L94

```
                case Branch: {
                    // Branch on constant -> jettison the not-taken block and merge.
                    if (isKnownDirection(block->cfaBranchDirection)) {
                        bool condition = branchCondition(block->cfaBranchDirection);
                        BasicBlock* targetBlock = block->successorForCondition(condition);
                        BasicBlock* jettisonedBlock = block->successorForCondition(!condition);
                        if (canMergeWithBlock(targetBlock)) {
                            if (extremeLogging)
                                m_graph.dump();
                            m_graph.dethread();
                            if (targetBlock == jettisonedBlock)
                                mergeBlocks(block, targetBlock, noBlocks());
                            else
                                mergeBlocks(block, targetBlock, oneBlock(jettisonedBlock)); // <---
add all of the nodes in tasrgetBlock to block.
```

Since it is always false, the branch direction is fixed. Merge false blocks.

7. DCE

> Global dead code elimination. Eliminates any node that is not NodeMustGenerate, not used by any other live node, and not subject to any type check.

Before and After

Unnecessary Check node is already deleted in CleanUp phase. In this phase, daed code and basic Phantom/Check clean-up.

```
  XX  0 2X:   D@60:< X:->       JSConstant(JS|UseAsOther, DoublePureNaN, Double: 9221120237041090560,
nan, bc#56, ExitValid)
  XX  0 2X:   D@61:<!0:->       MovHint(Check:Untyped:D@60, MustGen, loc11, W:SideState,
ClobbersExit, bc#56, ExitValid)
- 53  0 25:   D@62:< 1:->       SetLocal(Check:Untyped:D@60, loc11(T~<Double>/FlushedJSValue),
W:Stack(loc11), bc#56, exit: bc#64, ExitValid)  predicting DoublePureNaN

// DoubleRep and DoubleConstant are dead code because SameValue using D@90 and D@91 has been changed
to JSConstant.
// Therefore, DoubleRep and DoubleConstant are deleted.
// It just checks if value(D@28) is a RealNumber.
- 57  0 25:   D@90:< 1:->       DoubleRep(Check:RealNumber:D@28, Double|PureInt, BytecodeDouble,
Exits, bc#67, ExitValid)
+ 37  0 26:   D@90:<!0:->       Check(Check:RealNumber:D@28, MustGen, BytecodeDouble, Exits, bc#67,
ExitValid)
- 58  0 25:   D@91:< 1:->       DoubleConstant(Double|PureInt, BytecodeDouble, Double:
9221120237041090560, nan, bc#67, ExitValid)
  XX  0 2X:   D@68:< 1:->       JSConstant(Boolean|UseAsOther, Bool, False, bc#67, ExitValid)
  XX  0 2X:   D@69:<!0:->       MovHint(Check:Untyped:D@68, MustGen, loc7, W:SideState, ClobbersExit,
bc#67, ExitValid)
- 61  0 25:   D@74:< 1:->       SetLocal(Check:Untyped:D@68, loc7(V~<Boolean>/FlushedJSValue),
W:Stack(loc7), bc#67, exit: bc#73, ExitValid)  predicting Bool
```

Related code

/Source/JavaScriptCore/dfg/DFGDCEPhase.cpp # L115

```
        // This has to be a forward loop because we are using the insertion set.
        for (unsigned indexInBlock = 0; indexInBlock < block->size(); ++indexInBlock) {
            Node* node = block->at(indexInBlock);
            if (node->shouldGenerate())
                continue;

            if (node->flags() & NodeHasVarArgs) {
                ...
            }

            node->remove(m_graph);  // <--- convert node to Check.
            node->setRefCount(1);
        }
```

/Source/JavaScriptCore/dfg/DFGNode.cpp # L131

```
void Node::remove(Graph& graph)
{
    ...
    default:
        if (flags() & NodeHasVarArgs) {
            ...
        } else {
            children = children.justChecks();
            setOpAndDefaultFlags(Check);
        }
        return;
    }
```

/Source/JavaScriptCore/dfg/DFGDCEPhase.cpp # L65

```
            while (sourceIndex < block->size()) {
                Node* node = block->at(sourceIndex++);
                switch (node->op()) {
                case Check:
                case Phantom:
                    if (node->children.isEmpty()) // <--- Check converted from DoubleConstant is
dead.
                        continue;
                    break;                        // <--- Check converted from DoubleRep is alive.
                case CheckVarargs: {
                    bool isEmpty = true;
                    m_graph.doToChildren(node, [&] (Edge edge) {
                        isEmpty &= !edge;
                    });
                    if (isEmpty)
                        continue;
                    break;
                }
                default:
                    break;
                }
                block->at(targetIndex++) = node;
            }
```

Both DoubleRep and DoubleConstant were converted to Check nodes. Since DoubleConstant has empty children, the converted Check node is finally removed.

8. Graph after optimization!

Graph after DFG optimization

This is the final DFG graph:

```
Graph after optimization:

       29: DFG for object_is_opt#AATTmW:[0x1060f4360->0x1060f4240->0x106099700, DFGFunctionCall, 80]:
       29:   Fixpoint state: FixpointConverged; Form: ThreadedCPS; Unification state:
GloballyUnified; Ref count state: ExactRefCount
       29:   Arguments for block#0: D@0, D@1


  0  0 29:    D@0:< 2:->        SetArgumentDefinitely(IsFlushed, this(A~<Other>/FlushedJSValue),
machine:this, W:SideState, bc#0, ExitValid)  predicting Other
  1  0 29:    D@1:< 3:->        SetArgumentDefinitely(IsFlushed, arg1(B~<Double>/FlushedJSValue),
machine:arg1, W:SideState, bc#0, ExitValid)  predicting NonIntAsDouble
  2  0 29:    D@2:< 6:loc3>     JSConstant(JS|PureInt, Other, Undefined, bc#0, ExitValid)
  3  0 29:    D@3:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc0, W:SideState, ClobbersExit,
bc#0, ExitValid)
  4  0 29:    D@5:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc1, W:SideState, ClobbersExit,
bc#0, ExitInvalid)
  5  0 29:    D@7:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc2, W:SideState, ClobbersExit,
bc#0, ExitInvalid)
  6  0 29:    D@9:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc3, W:SideState, ClobbersExit,
bc#0, ExitInvalid)
  7  0 29:   D@11:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc4, W:SideState, ClobbersExit,
bc#0, ExitInvalid)
  8  0 29:   D@13:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc5, W:SideState, ClobbersExit,
bc#0, ExitInvalid)


  9  0 29:   D@16:< 3:loc3>     JSConstant(JS|UseAsOther, OtherObj, Weak:Object: 0x104034428 with
butterfly 0x0(base=0xfffffffffffffff8) (Structure %CD:JSGlobalLexicalEnvironment), StructureID:
21424, bc#1, ExitValid)
 10  0 29:   D@17:<!0:->        MovHint(Check:Untyped:D@16, MustGen, loc4, W:SideState, ClobbersExit,
bc#1, ExitValid)


 11  0 29:   D@19:<!0:->        MovHint(Check:Untyped:D@16, MustGen, loc5, W:SideState, ClobbersExit,
bc#3, ExitValid)
 12  0 29:   D@21:<!0:->        InvalidationPoint(MustGen, W:SideState, Exits, bc#6, ExitValid)
 13  0 29:   D@22:< 1:loc4>     JSConstant(JS|PureInt, Empty, <JSValue()>, bc#7, ExitValid)
 14  0 29:   D@23:<!0:->        MovHint(Check:Untyped:D@22, MustGen, loc6, W:SideState, ClobbersExit,
bc#7, ExitValid)


 15  0 29:   D@25:< 7:loc4>     NewObject(JS|UseAsOther, Final, %Db:Object, R:HeapObjectCount,
W:HeapObjectCount, Exits, bc#10, ExitValid)
 16  0 29:   D@26:<!0:->        MovHint(Check:Untyped:D@25, MustGen, loc7, W:SideState, ClobbersExit,
bc#10, ExitValid)
 17  0 29:   D@28:<!4:loc5>     GetLocal(Check:Untyped:D@1, JS|MustGen|UseAsOther, NonIntAsDouble,
arg1(B~<Double>/FlushedJSValue), machine:arg1, R:Stack(arg1), bc#14, ExitValid)  predicting
NonIntAsDouble
 18  0 29:   D@29:<!0:->        FilterPutByStatus(Check:Untyped:D@25, MustGen, (<id='uid:(p0)',
Transition: [0x300008bd0:[0x8bd0/35792, Object, (0/2, 0/0){}, NonArray, Proto:0x104011968]] to
0x300008c40:[0x8c40/35904, Object, (1/2, 0/0){p0:0}, NonArray, Proto:0x104011968, Leaf], [[]], offset
= 0, >), W:SideState, bc#14, ExitValid)
 19  0 29:   D@89:<!0:->        Check(Check:Number:D@28, MustGen, Exits, bc#14, ExitValid)
 20  0 29:   D@80:<!0:->        Phantom(Check:Untyped:D@16, MustGen, bc#14, ExitValid)
 21  0 29:   D@31:<!0:->        PutByOffset(KnownCell:D@25, KnownCell:D@25, Check:Untyped:D@28,
MustGen, id0{p0}, 0, W:NamedProperties(0), ClobbersExit, bc#14, ExitValid)
 22  0 29:   D@32:<!0:->        PutStructure(KnownCell:D@25, MustGen, %Db:Object -> %Cd:Object,
ID:35904, R:JSObject_butterfly, W:JSCell_indexingType,JSCell_structureID,JSCell_typeInfoFlags,
ClobbersExit, bc#14, ExitInvalid)


 23  0 29:   D@33:<!0:->        MovHint(Check:Untyped:D@25, MustGen, loc6, W:SideState, ClobbersExit,
bc#20, ExitValid)
 24  0 29:   D@35:< 2:loc3>     JSConstant(JS|UseAsOther, OtherObj, Weak:Object: 0x1060c0068 with
butterfly 0x7001014408(base=0x7001014000) (Structure %DJ:global), StructureID: 33440, bc#23,
ExitValid)
```

```
 25  0 29:   D@36:<!0:->        MovHint(Check:Untyped:D@35, MustGen, loc10, W:SideState,
ClobbersExit, bc#23, ExitValid)
 26  0 29:   D@41:< 4:loc6>     JSConstant(JS|UseAsOther, OtherObj, Weak:Object: 0x104038c88 with
butterfly 0x7001008448(base=0x70010083c0) (Structure %B7:Function), StructureID: 36016, bc#30,
ExitValid)
 27  0 29:   D@43:<!0:->        MovHint(Check:Untyped:D@41, MustGen, loc11, W:SideState,
ClobbersExit, bc#30, ExitValid)
 28  0 29:   D@45:<!0:->        MovHint(Check:Untyped:D@41, MustGen, loc10, W:SideState,
ClobbersExit, bc#38, ExitValid)


 29  0 29:   D@47:<!0:->        FilterGetByStatus(Check:Untyped:D@41, MustGen, (Simple, <id='uid:
(is)', [0x300008cb0:[0x8cb0/36016, Function, (0/0, 14/16){length:64, name:65, prototype:66,
getPrototypeOf:67, getOwnPropertyDescriptor:68, getOwnPropertyNames:69, getOwnPropertySymbols:70,
keys:71, defineProperty:72, create:73, values:74, hasOwn:75, hasOwn:76, is:77}, NonArray,
Proto:0x104038508, Leaf]], [], offset = 77>, seenInJIT = true), W:SideState, bc#41, ExitValid)
 30  0 29:   D@50:< 3:loc7>     JSConstant(JS|UseAsOther, Function, Weak:Object: 0x10608dec0 with
butterfly 0x7001001fe8(base=0x7001001fc0) (Structure %Dh:Function), StructureID: 22432, bc#41,
ExitValid)
 31  0 29:   D@51:<!0:->        MovHint(Check:Untyped:D@50, MustGen, loc7, W:SideState, ClobbersExit,
bc#41, ExitValid)
 32  0 29:   D@53:<!0:->        MovHint(Check:Untyped:D@28, MustGen, loc9, W:SideState, ClobbersExit,
bc#46, ExitValid)
 33  0 29:   D@56:<!0:->        MovHint(Check:Untyped:D@35, MustGen, loc8, W:SideState, ClobbersExit,
bc#49, ExitValid)


 34  0 29:   D@60:< 3:loc3>     JSConstant(JS|UseAsOther, DoublePureNaN, Double: 9221120237041090560,
nan, bc#56, ExitValid)
 35  0 29:   D@61:<!0:->        MovHint(Check:Untyped:D@60, MustGen, loc11, W:SideState,
ClobbersExit, bc#56, ExitValid)
 36  0 29:   D@63:<!0:->        MovHint(Check:Untyped:D@60, MustGen, loc8, W:SideState, ClobbersExit,
bc#64, ExitValid)


 37  0 29:   D@65:<!0:->        FilterCallLinkStatus(Check:Untyped:D@50, MustGen, Statically Proved,
(Function: Object: 0x10608dec0 with butterfly 0x7001001fe8(base=0x7001001fc0) (Structure 0x3000057a0:
[0x57a0/22432, Function, (0/0, 2/4){length:64, name:65}, NonArray, Proto:0x104038508, Leaf]),
StructureID: 22432; Executable: NativeExecutable:0x10b5aca1c/0x10b3bea14), W:SideState, bc#67,
ExitValid)
 38  0 29:   D@90:<!0:->        Check(Check:RealNumber:D@28, MustGen, BytecodeDouble, Exits, bc#67,
ExitValid)
 39  0 29:   D@75:<!0:->        Phantom(Check:Untyped:D@50, MustGen, bc#67, ExitValid)
 40  0 29:   D@73:<!0:->        Phantom(Check:Untyped:D@60, MustGen, bc#67, ExitValid)
 41  0 29:   D@72:<!0:->        Phantom(Check:Untyped:D@41, MustGen, bc#67, ExitValid)
 42  0 29:   D@68:< 1:loc6>     JSConstant(Boolean|UseAsOther, Bool, False, bc#67, ExitValid)
 43  0 29:   D@69:<!0:->        MovHint(Check:Untyped:D@68, MustGen, loc7, W:SideState, ClobbersExit,
bc#67, ExitValid)


 44  0 29:   D@92:<!0:->        Flush(Check:Untyped:D@0, MustGen|IsFlushed, this(A~
<Other>/FlushedJSValue), machine:this, R:Stack(this), W:SideState, bc#73, ExitValid)  predicting
Other
 45  0 29:   D@93:<!0:->        Flush(Check:Untyped:D@1, MustGen|IsFlushed, arg1(B~
<Double>/FlushedJSValue), machine:arg1, R:Stack(arg1), W:SideState, bc#73, ExitValid)  predicting
NonIntAsDouble
 46  0 29:   D@84:<!0:->        CheckTierUpAtReturn(MustGen, W:SideState, Exits, bc#78, ExitValid)


 47  0 29:   D@81:<!0:->        Return(Check:Untyped:D@25, MustGen, W:SideState, Exits, bc#78,
ExitValid)
 48  0 29:   D@82:<!0:->        Flush(Check:Untyped:D@1, MustGen|IsFlushed, arg1(B~
<Double>/FlushedJSValue), machine:arg1, R:Stack(arg1), W:SideState, bc#78, ExitValid)  predicting
NonIntAsDouble
 49  0 29:   D@83:<!0:->        Flush(Check:Untyped:D@0, MustGen|IsFlushed, this(A~
<Other>/FlushedJSValue), machine:this, R:Stack(this), W:SideState, bc#78, ExitValid)  predicting
Other
```

Graph after FTL optimization

FTL optimization has the same phases as DFG and more advanced phases.

```
Graph just before FTL lowering:

       56: DFG for object_is_opt#AATTmW:[0x1040f4480->0x1040f4240->0x104099700, DFGFunctionCall, 80
(DidTryToEnterInLoop)]:
       56:   Fixpoint state: FixpointConverged; Form: SSA; Unification state: GloballyUnified; Ref
count state: ExactRefCount
       56:   Argument formats for entrypoint index: 0 : FlushedJSValue, FlushedJSValue

  0  0 56:   D@16:< 2:->        JSConstant(JS|UseAsOther, OtherObj, Weak:Object: 0x103034428 with
butterfly 0x0(base=0xfffffffffffffff8) (Structure %DV:JSGlobalLexicalEnvironment), StructureID:
21424, bc#0, ExitValid)
  1  0 56:   D@60:< 2:->        JSConstant(JS|UseAsOther, DoublePureNaN, Double: 9221120237041090560,
nan, bc#0, ExitValid)
  2  0 56:   D@35:< 2:->        JSConstant(JS|UseAsOther, OtherObj, Weak:Object: 0x1040c0068 with
butterfly 0x700f014408(base=0x700f014000) (Structure %DQ:global), StructureID: 33440, bc#0,
ExitValid)
  3  0 56:    D@2:< 6:->        JSConstant(JS|PureInt, Other, Undefined, bc#0, ExitValid)
  4  0 56:   D@50:< 2:->        JSConstant(JS|UseAsOther, Function, Weak:Object: 0x10408dec0 with
butterfly 0x700f001fe8(base=0x700f001fc0) (Structure %Ae:Function), StructureID: 22432, bc#0,
ExitValid)
  5  0 56:   D@54:< 1:->        JSConstant(Boolean|UseAsOther, Bool, False, bc#0, ExitValid)
  6  0 56:   D@41:< 3:->        JSConstant(JS|UseAsOther, OtherObj, Weak:Object: 0x103038c88 with
butterfly 0x700f008448(base=0x700f0083c0) (Structure %DVl:Function), StructureID: 36016, bc#0,
ExitValid)
  7  0 56:   D@22:< 1:->        JSConstant(JS|PureInt, Empty, <JSValue()>, bc#0, ExitValid)
  8  0 56:   D@44:<!0:->        ExitOK(MustGen, W:SideState, bc#0, ExitValid)
  9  0 56:   D@14:<!0:->        InitializeEntrypointArguments(MustGen, W:SideState, ClobbersExit,
bc#0, ExitValid)
 10  0 56:   D@28:<!0:->        ExitOK(MustGen, W:SideState, bc#0, ExitValid)
 11  0 56:   D@38:< 4:->        GetStack(JS|PureInt, NonIntAsDouble, arg1, machine:arg1,
FlushedJSValue, R:Stack(arg1), bc#0, ExitValid)
 12  0 56:   D@46:<!0:->        KillStack(MustGen, loc0, W:Stack(loc0), ClobbersExit, bc#0,
ExitValid)
 13  0 56:    D@3:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc0, W:SideState, ClobbersExit,
bc#0, ExitInvalid)
 14  0 56:   D@58:<!0:->        KillStack(MustGen, loc1, W:Stack(loc1), ClobbersExit, bc#0,
ExitInvalid)
 15  0 56:    D@5:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc1, W:SideState, ClobbersExit,
bc#0, ExitInvalid)
 16  0 56:   D@62:<!0:->        KillStack(MustGen, loc2, W:Stack(loc2), ClobbersExit, bc#0,
ExitInvalid)
 17  0 56:    D@7:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc2, W:SideState, ClobbersExit,
bc#0, ExitInvalid)
 18  0 56:   D@59:<!0:->        KillStack(MustGen, loc3, W:Stack(loc3), ClobbersExit, bc#0,
ExitInvalid)
 19  0 56:    D@9:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc3, W:SideState, ClobbersExit,
bc#0, ExitInvalid)
 20  0 56:   D@57:<!0:->        KillStack(MustGen, loc4, W:Stack(loc4), ClobbersExit, bc#0,
ExitInvalid)
 21  0 56:   D@11:<!0:->        MovHint(Check:Untyped:D@2, MustGen, loc4, W:SideState, ClobbersExit,
bc#0, ExitInvalid)
 22  0 56:   D@55:<!0:->        KillStack(MustGen, loc5, W:Stack(loc5), ClobbersExit, bc#0,
ExitInvalid)
 23  0 56:   D@13:<!0:->        MovHint(Check:Untyped:Kill:D@2, MustGen, loc5, W:SideState,
ClobbersExit, bc#0, ExitInvalid)


 24  0 56:   D@49:<!0:->        KillStack(MustGen, loc4, W:Stack(loc4), ClobbersExit, bc#1,
ExitValid)
 25  0 56:   D@17:<!0:->        MovHint(Check:Untyped:D@16, MustGen, loc4, W:SideState, ClobbersExit,
bc#1, ExitInvalid)
 26  0 56:   D@48:<!0:->        KillStack(MustGen, loc5, W:Stack(loc5), ClobbersExit, bc#3,
ExitValid)
```

```
 27  0 56:   D@19:<!0:->        MovHint(Check:Untyped:Kill:D@16, MustGen, loc5, W:SideState,
ClobbersExit, bc#3, ExitInvalid)
 28  0 56:   D@21:<!0:->        InvalidationPoint(MustGen, W:SideState, Exits, bc#6, ExitValid)
 29  0 56:   D@42:<!0:->        KillStack(MustGen, loc6, W:Stack(loc6), ClobbersExit, bc#7,
ExitValid)
 30  0 56:   D@23:<!0:->        MovHint(Check:Untyped:Kill:D@22, MustGen, loc6, W:SideState,
ClobbersExit, bc#7, ExitInvalid)


 31  0 56:   D@25:< 7:->        NewObject(JS|UseAsOther, Final, %AA:Object, R:HeapObjectCount,
W:HeapObjectCount, Exits, bc#10, ExitValid)
 32  0 56:   D@40:<!0:->        KillStack(MustGen, loc7, W:Stack(loc7), ClobbersExit, bc#10,
ExitValid)
 33  0 56:   D@26:<!0:->        MovHint(Check:Untyped:D@25, MustGen, loc7, W:SideState, ClobbersExit,
bc#10, ExitInvalid)
 34  0 56:   D@29:<!0:->        FilterPutByStatus(Check:Untyped:D@25, MustGen, (<id='uid:(p0)',
Transition: [0x300008bd0:[0x8bd0/35792, Object, (0/2, 0/0){}, NonArray, Proto:0x103011968]] to
0x300008c40:[0x8c40/35904, Object, (1/2, 0/0){p0:0}, NonArray, Proto:0x103011968, Leaf (Watched)],
[[]], offset = 0, >), W:SideState, bc#14, ExitValid)
 35  0 56:    D@4:<!0:->        Check(Check:Number:D@38, MustGen, Exits, bc#14, ExitValid)
 36  0 56:   D@31:<!0:->        PutByOffset(KnownCell:D@25, KnownCell:D@25, Check:Untyped:D@38,
MustGen, id0{p0}, 0, W:NamedProperties(0), ClobbersExit, bc#14, ExitValid)
 37  0 56:   D@32:<!0:->        PutStructure(KnownCell:D@25, MustGen, %AA:Object -> %DA:Object,
ID:35904, R:JSObject_butterfly, W:JSCell_indexingType,JSCell_structureID,JSCell_typeInfoFlags,
ClobbersExit, bc#14, ExitInvalid)


 38  0 56:   D@39:<!0:->        KillStack(MustGen, loc6, W:Stack(loc6), ClobbersExit, bc#20,
ExitValid)
 39  0 56:   D@33:<!0:->        MovHint(Check:Untyped:D@25, MustGen, loc6, W:SideState, ClobbersExit,
bc#20, ExitInvalid)
 40  0 56:   D@37:<!0:->        KillStack(MustGen, loc10, W:Stack(loc10), ClobbersExit, bc#23,
ExitValid)
 41  0 56:   D@36:<!0:->        MovHint(Check:Untyped:D@35, MustGen, loc10, W:SideState,
ClobbersExit, bc#23, ExitInvalid)
 42  0 56:   D@30:<!0:->        KillStack(MustGen, loc11, W:Stack(loc11), ClobbersExit, bc#30,
ExitValid)
 43  0 56:   D@43:<!0:->        MovHint(Check:Untyped:D@41, MustGen, loc11, W:SideState,
ClobbersExit, bc#30, ExitInvalid)
 44  0 56:   D@10:<!0:->        KillStack(MustGen, loc10, W:Stack(loc10), ClobbersExit, bc#38,
ExitValid)
 45  0 56:   D@45:<!0:->        MovHint(Check:Untyped:D@41, MustGen, loc10, W:SideState,
ClobbersExit, bc#38, ExitInvalid)


 46  0 56:   D@47:<!0:->        FilterGetByStatus(Check:Untyped:Kill:D@41, MustGen, (Simple,
<id='uid:(is)', [0x300008cb0:[0x8cb0/36016, Function, (0/0, 14/16){length:64, name:65, prototype:66,
getPrototypeOf:67, getOwnPropertyDescriptor:68, getOwnPropertyNames:69, getOwnPropertySymbols:70,
keys:71, defineProperty:72, create:73, values:74, hasOwn:75, hasOwn:76, is:77}, NonArray,
Proto:0x103038508, Leaf (Watched)]], [], offset = 77>, seenInJIT = true), W:SideState, bc#41,
ExitValid)
 47  0 56:    D@8:<!0:->        KillStack(MustGen, loc7, W:Stack(loc7), ClobbersExit, bc#41,
ExitValid)
 48  0 56:   D@51:<!0:->        MovHint(Check:Untyped:D@50, MustGen, loc7, W:SideState, ClobbersExit,
bc#41, ExitInvalid)
 49  0 56:    D@6:<!0:->        KillStack(MustGen, loc9, W:Stack(loc9), ClobbersExit, bc#46,
ExitValid)
 50  0 56:   D@53:<!0:->        MovHint(Check:Untyped:D@38, MustGen, loc9, W:SideState, ClobbersExit,
bc#46, ExitInvalid)
 51  0 56:   D@18:<!0:->        KillStack(MustGen, loc8, W:Stack(loc8), ClobbersExit, bc#49,
ExitValid)
 52  0 56:   D@56:<!0:->        MovHint(Check:Untyped:Kill:D@35, MustGen, loc8, W:SideState,
ClobbersExit, bc#49, ExitInvalid)
 53  0 56:   D@20:<!0:->        KillStack(MustGen, loc11, W:Stack(loc11), ClobbersExit, bc#56,
ExitValid)
 54  0 56:   D@61:<!0:->        MovHint(Check:Untyped:D@60, MustGen, loc11, W:SideState,
ClobbersExit, bc#56, ExitInvalid)
```

```
 55  0 56:   D@24:<!0:->        KillStack(MustGen, loc8, W:Stack(loc8), ClobbersExit, bc#64,
ExitValid)
 56  0 56:   D@63:<!0:->        MovHint(Check:Untyped:Kill:D@60, MustGen, loc8, W:SideState,
ClobbersExit, bc#64, ExitInvalid)


 57  0 56:   D@64:<!0:->        FilterCallLinkStatus(Check:Untyped:Kill:D@50, MustGen, Statically
Proved, (Function: Object: 0x10408dec0 with butterfly 0x700f001fe8(base=0x700f001fc0) (Structure
0x3000057a0:[0x57a0/22432, Function, (0/0, 2/4){length:64, name:65}, NonArray, Proto:0x103038508,
Leaf (Watched)]), StructureID: 22432; Executable: NativeExecutable:0x10b5aca1c/0x10b3bea14),
W:SideState, bc#67, ExitValid)
 58  0 56:    D@1:<!0:->        Check(Check:RealNumber:Kill:D@38, MustGen, BytecodeDouble, Exits,
bc#67, ExitValid)
 59  0 56:   D@27:<!0:->        KillStack(MustGen, loc7, W:Stack(loc7), ClobbersExit, bc#67,
ExitValid)
 60  0 56:   D@52:<!0:->        MovHint(Check:Untyped:Kill:D@54, MustGen, loc7, W:SideState,
ClobbersExit, bc#67, ExitInvalid)
 61  0 56:   D@12:<!0:->        Return(Check:Untyped:Kill:D@25, MustGen, W:SideState, Exits, bc#78,
ExitValid)
```

The code of interest is this:

```
// D@38 is value
 11  0 56:   D@38:< 4:->        GetStack(JS|PureInt, NonIntAsDouble, arg1, machine:arg1,
FlushedJSValue, R:Stack(arg1), bc#0, ExitValid)


// D@25 is tmp
 31  0 56:   D@25:< 7:->        NewObject(JS|UseAsOther, Final, %AA:Object, R:HeapObjectCount,
W:HeapObjectCount, Exits, bc#10, ExitValid)


// If value is RealNumber, then return tmp
 58  0 56:    D@1:<!0:->        Check(Check:RealNumber:Kill:D@38, MustGen, BytecodeDouble, Exits,
bc#67, ExitValid)
 59  0 56:   D@27:<!0:->        KillStack(MustGen, loc7, W:Stack(loc7), ClobbersExit, bc#67,
ExitValid)
 60  0 56:   D@52:<!0:->        MovHint(Check:Untyped:Kill:D@54, MustGen, loc7, W:SideState,
ClobbersExit, bc#67, ExitInvalid)
 61  0 56:   D@12:<!0:->        Return(Check:Untyped:Kill:D@25, MustGen, W:SideState, Exits, bc#78,
ExitValid)
```

The complex code has been optimized to be simple, returning tmp if value is a RealNumber rather than NaN.

9. B3 Lowering

FTL lower DFG to B3. The optimized nodes so far are lowered through compilation.

Related code

/Source/JavaScriptCore/ftl/FTLLowerDFGToB3.cpp # L699

```
bool compileNode(unsigned nodeIndex)
{
    ...
    switch (m_node->op()) {
    ...
    case DFG::Check:
    case CheckVarargs:
        compileNoOp();
        break;
```

/Source/JavaScriptCore/ftl/FTLLowerDFGToB3.cpp # L2190

```
void compileNoOp()
{
```

```
        DFG_NODE_DO_TO_CHILDREN(m_graph, m_node, speculate);
    }
```

/Source/JavaScriptCore/ftl/FTLLowerDFGToB3.cpp # L19322

```
    void speculate(Edge edge)
    {
        switch (edge.useKind()) {
        ...
        case RealNumberUse:
            speculateRealNumber(edge);
            break;
```

`DFG::Check` node speculate to children. And it calls the `speculateRealNumber` function where the vulnerability occurs.

/Source/JavaScriptCore/ftl/FTLLowerDFGToB3.cpp # L20268

```
    void speculateRealNumber(Edge edge)
    {
        // Do an early return here because lowDouble() can create a lot of control flow.
        if (!m_interpreter.needsTypeCheck(edge))
            return;

        LValue value = lowJSValue(edge, ManualOperandSpeculation);
        LValue doubleValue = unboxDouble(value);

        LBasicBlock intCase = m_out.newBlock();
        LBasicBlock continuation = m_out.newBlock();

        m_out.branch(
            m_out.doubleEqual(doubleValue, doubleValue),
            usually(continuation), rarely(intCase));

        LBasicBlock lastNext = m_out.appendTo(intCase, continuation);

        typeCheck(
            jsValueValue(value), m_node->child1(), SpecBytecodeRealNumber,
            isNotInt32(value, provenType(m_node->child1()) & ~SpecFullDouble));
        m_out.jump(continuation);

        m_out.appendTo(continuation, lastNext);
    }
```

B3 graph

This is the key part of the lowered graph:

```
b3      Int64 b@12 = FramePointer()


b3      Int64 b@14 = Const64(-562949953421312)


b3      Int64 b@62 = Const64(48, D@38)
b3      Int64 b@63 = Add(b@12, $48(b@62), D@38)
b3      Int64 b@64 = Load(b@63, ControlDependent|Reads:130, D@38)


b3      Int64 b@246 = Add(b@64, $-562949953421312(b@14), D@1)
b3      Double b@247 = BitwiseCast(b@246, D@1)
b3      Int32 b@248 = Equal(b@247, b@247, D@1)
b3      Void b@249 = Branch(b@248, Terminal, D@1)
```

This is key part of the final B3 graph:

```
b3      Int64 b@12 = FramePointer()
b3      Int64 b@64 = Load(b@12, offset = 48, ControlDependent|Reads:130, D@38)


b3      Int64 b@14 = Const64(-562949953421312)
b3      Int64 b@169 = BitAnd(b@64, $-562949953421312(b@14), D@4)
b3      Int32 b@170 = Equal(b@169, $0(b@2), D@4)
b3      Void b@171 = Check(b@170:WarmAny, b@64:ColdAny, b@197:ColdAny, generator = 0x102043c90,
earlyClobbered = [], lateClobbered = [], usedRegisters = [], ExitsSideways|Reads:Top, D@4)
```

It compiles to the following machine code:

```
asm                         <68> 0x12002c044:    ldur     x5, [fp, #48]

asm                        <240> 0x12002c0f0:    tst      x5, #0xfffe000000000000 ; bit AND
asm                        <244> 0x12002c0f4:    b.eq     0x12002c1e0 -> <480>    ; branch if Z == 1
asm                        <248> 0x12002c0f8:    stur     x5, [x0, #16]           ; tmp = {p0: value}
asm                        <252> 0x12002c0fc:    movz     x1, #0x8c40 -> 35904    ; structure ID
asm                        <256> 0x12002c100:    stur     w1, [x0]

asm                        <272> 0x12002c110:    mov      sp, fp
asm                        <276> 0x12002c114:    ldp      fp, lr, [sp], #16
asm                        <280> 0x12002c118:    ret      lr
```

According to JSValue, Even when comparing NaN, the `tst` result always does not set the Z flag unless x5 is a Pointer.

> Pointer { 0000:PPPP:PPPP:PPPP
> / 0001:****:****:****
> Double { …
> \ FFFE:****:****:****
> Integer { FFFF:0000:IIII:IIII

Is DFG's speculativeRealNumber safe?

The vulnerability is in FTL's speculativeRealNumber. Then, how is DFG's speculativeRealNumber different?

/Source/JavaScriptCore/dfg/DFGSpeculativeJIT.cpp # L11703

```
void SpeculativeJIT::speculateRealNumber(Edge edge)
{
    if (!needsTypeCheck(edge, SpecBytecodeRealNumber))
        return;

    JSValueOperand op1(this, edge, ManualOperandSpeculation);
    FPRTemporary result(this);

    JSValueRegs op1Regs = op1.jsValueRegs();
    FPRReg resultFPR = result.fpr();

#if USE(JSVALUE64)
    GPRTemporary temp(this);
    GPRReg tempGPR = temp.gpr();
    m_jit.unboxDoubleWithoutAssertions(op1Regs.gpr(), tempGPR, resultFPR);
#else
    unboxDouble(op1Regs.tagGPR(), op1Regs.payloadGPR(), resultFPR);
#endif

    JITCompiler::Jump done = m_jit.branchIfNotNaN(resultFPR);     // <--- How to check NaN ?

    typeCheck(op1Regs, edge, SpecBytecodeRealNumber, m_jit.branchIfNotInt32(op1Regs));
```

```
    done.link(&m_jit);
}
```

/Source/JavaScriptCore/jit/AssemblyHelpers.h # L1161

```
    Jump branchIfNotNaN(FPRReg fpr)
    {
        return branchDouble(DoubleEqualAndOrdered, fpr, fpr);     // <--- It uses DoubleEqual.
    }
```

/Source/JavaScriptCore/assembler/MacroAssemblerARM64.h # L2349

```
    Jump branchDouble(DoubleCondition cond, FPRegisterID left, FPRegisterID right)
    {
        m_assembler.fcmp<64>(left, right);
        return jumpAfterFloatingPointCompare(cond);               // <--- It uses floating point
comparison, not equal.
    }
```

/Source/JavaScriptCore/assembler/ARM64Assembler.h # L2140

```
    template<int datasize>
    ALWAYS_INLINE void fcmp(FPRegisterID vn, FPRegisterID vm)
    {
        CHECK_DATASIZE();
        insn(floatingPointCompare(DATASIZE, vm, vn, FPCmpOp_FCMP));
    }
    ...
    ALWAYS_INLINE static int floatingPointCompare(Datasize type, FPRegisterID rm, FPRegisterID rn,
FPCmpOp opcode2)
    {
        const int M = 0;
        const int S = 0;
        const int op = 0;
        return (0x1e202000 | M << 31 | S << 29 | type << 22 | rm << 16 | op << 14 | rn << 5 |
opcode2);
    }
```

The FTL JIT reuses the DFG's compiler pipeline and adds new optimizations. FTL and DFG share code whenever possible, but sometimes have different definitions of the same operation.

This is part 1, part 2 will describe how to abuse this jit bug to make AAR/AAW primitives.

REF

Instruction

- http://www.nacad.ufrj.br/online/intel/vtune/users_guide/mergedProjects/analyzer_ec/mergedProjects/refe
- https://wiki.osdev.org/CPU_Registers_x86-64#RFLAGS_Register
- https://faydoc.tripod.com/cpu/setnp.htm

JIT

- https://webkit.org/blog/10308/speculation-in-javascriptcore/
- https://webkit.org/blog/3362/introducing-the-webkit-ftl-jit/