Intro

The debugger partly relies on patching ICs to set break points. The implementation for this is complex and error-prone. I'd like to remove this to reduce complexity. I'm sure this will also improve productivity in the long run, as a number of team members already experienced problems with this part of code whenever full codegen was changed.

Current state

In order to set a debug break point in a function, we have three ways:

- The code has been previously recompiled for debugging. This means that the code contains debug break slots at expression and statement positions. Debug break slots are a sequence of NOP instructions that can be overwritten to call into the debugger.
- The code contains ICs that can be overwritten by debug versions of the original ICs. The debug versions call into the debugger first, and then return via trampoline code to the original IC that carries out the actual work.
- Return statements (explicit and implicit ones) can be patched to jump into the debugger first.

Up until now, we must rely on debug break ICs in situations where we cannot recompile code for debugging, so debug break slots are simply not available. Setting break points in such code is a bit unpredictable, since some break locations don't exist.

This situation has been fixed in a series of CLs. We now can guarantee that code in which we want to set break points, have been recompiled to include debug break slots.

Motivation

Having both debug break slots and debug break ICs has several disadvantages.

- Statements may contain ICs. In order to avoid having both debug break slots and IC for the same statement and source position, we have an elaborate system to predict whether a debug break slot should be skipped in favor of an IC (BreakableStatementChecker). However, this is often not up-to-date and leads to bugs.
- Some statements are implemented as branches where one branch is implemented by IC and the other is not (e.g. assignment to a variable that could be context-allocated or on the global object). The algorithm looking for break locations for a given position does not expect this and only returns one. In this case, we may either hit no break point at all or multiple break points with the same source position in succession (when stepping). To work around the latter situation, the debugger ignores multiple breaks on the same

source position. That is also wrong, for example in an empty infinite while loop (“while(true) {}”).

- We keep a copy of unpatched code around so that we can return to the original IC after returning from the call to the debugger, and also to restore the original IC.
- ICs need to be cleared to prepare for setting the break point.

Approach

To incrementally migrate to only using debug break slots, I will step-by-step change BreakLocation::Iterator to skip types of ICs and replace occurrences of those ICs in full codegen by statement positions. In order to have those statement positions actually emit debug break slots, BreakableStatementChecker will have to be changed as well.

This will probably cause some minor changes to debug break locations that should be benign wrt. user experience, but may cause minor test expectation changes, so each failing test must be inspected to decide whether to rebaseline.

Eventually, I can rip out

- Whole machinery of setting and patching ICs for debugging
- Keeping a copy of the original code on the DebugInfo object
- BreakableStatementChecker
- Test cases that specifically check for ICs at certain source positions for debugging