#### LLVM

The LLVM compiler infrastructure project is a collection of modular and reusable compiler and toolchain technologies used to develop compiler front ends and back ends. At its heart, LLVM is a library for programmatically creating machine-native code. A developer uses the API to generate instructions in a format called an *intermediate representation*, or IR. LLVM can then compile the IR into a standalone binary, or perform a JIT (just-in-time) compilation on the code to run in the context of another program, such as an interpreter for the language.

#### Resources to better understand the system

* LLVM Architecture (<http://www.aosabook.org/en/llvm.html>)
* LLVM’s analysis and transform passes (<https://llvm.org/docs/Passes.html>)
* What is LLVM ? (<https://www.infoworld.com/article/3247799/development-tools/what-is-llvm-the-power-behind-swift-rust-clang-and-more.html>)

#### Variation points

time\_passes || gvn|| licm || instcombine || inline

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Variation Point | Range | Description |
| 1 | time\_passes | On|Off | Record the amount of time needed for each pass and print it to standard error. |
| 2 | gvn | On|Off | Optimizations are implemented as Passes that traverse some portion of a program to either collect information or transform the program.  GVN is a transform pass ( mutates the program in some way) that performs global value numbering to eliminate fully and partially redundant instructions. It also performs redundant load (Load instruction) elimination.  Loads can be quite expensive (e.g. if they miss in the cache), so advantageous to eliminate redundant loads.  (Older:<http://blog.llvm.org/2009/12/introduction-to-load-elimination-in-gvn.html>)  A primary limitations of gvn is it is slow for large test cases (<http://lists.llvm.org/pipermail/llvm-dev/2016-November/107110.htm>l) |
| 3 | licm | On|Off | This is a LLVM transform pass that performs loop invariant code motion, attempting to remove as much code from the body of a loop as possible.  This transformation identifies expressions that are unnecessarily re-evaluated at successive loop iterations (loop-invariant expressions), and subsequently moves them to a program point where they execute fewer times. LICM is an attractive optimization because it almost invariably achieves a speedup in execution time, due to reducing the number of times certain statements are evaluated.  Because of their generality, traditional compiler optimizations may achieve suboptimal results in application domains where programs adhere to a very particular, perhaps unusual structure. In such cases, specifically tailored transformations often produce superior code compared to even the most successful general-purpose compilers.  (<https://www.doc.ic.ac.uk/teaching/distinguished-projects/2015/p.colea.pdf>) |
| 4 | instcombine | On|Off | Combine instructions to form fewer, simple instructions.  Advantage of eliminating unneeded instructions.  Disadvantage: InstCombine known to be buggy. <https://llvm.org/devmtg/2014-10/Slides/Menendez-Alive.pdf> |
| 5 | inline | On|Off | Bottom-up inlining of functions into callees. Has performance to (code) size tradeoffs. Embedded targets in general are memory constrained but need real time performance.  Inlining cost calculation is also impacted by several factors: Frequency vs latency, amount of inling etc.  (<https://llvm.org/devmtg/2017-02-04/Impact-of-the-current-LLVM-inlining-strategy.pdf>) |