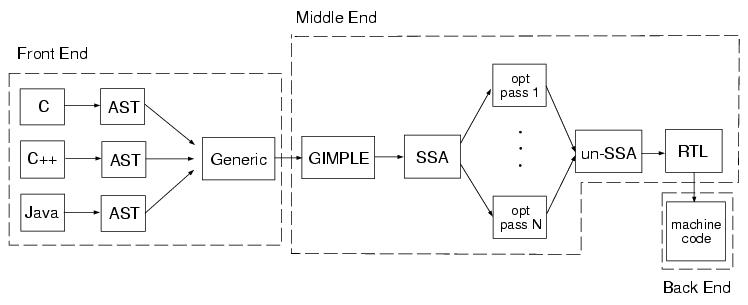
**An Overview of GCC Architecture**

# Introduction:

GNU Compiler Collection (GCC) comprises a number of compilers for different programming languages. The main GCC executable **gcc** processes source files written in C, C++, Objective-C, Objective-C++, Java, Fortran, or Ada and produces an assembly file for each source file. It is a driver program that invokes the appropriate compilation programs depending on the language of the source file. For a C source file they are the preprocessor and compiler **cc1**, the assembler **as**, and the linker **collect2**. The first and the third programs come with a GCC distribution, the assembler is a part of the GNU binutils package. This book describes the internals of the preprocessor and compiler **cc1**.

# Structure:

Each compiler includes the following three components: a front end, a middle end, and a back end. GCC compiles one file at a time. A source file goes through all three components one after another. [Figure 1](https://en.wikibooks.org/wiki/File:Gcc.JPG) illustrates the components and the source file representations associated with each component.



The purpose of the front end is to read the source file, parse it, and convert it into the standard *abstract syntax tree (AST)* representation. There is one front end for each programming language. Because of the differences in languages, the format of the generated ASTs is slightly different for each language. The next step after AST generation is the unification step in which the AST tree is converted into a unified form called generic. After this, the middle end part of the compiler takes control. First, the tree is converted into another representation called *GIMPLE*. In this form, each expression contains no more than three operands, all control flow constructs are represented as combinations of conditional statements and goto operators, arguments of a function call can only be variables, etc. [Figure 2](https://en.wikibooks.org/w/index.php?title=File:Gccinternals-img2.png&action=edit&redlink=1) illustrates the differences between a tree in generic form and a tree in GIMPLE form. GIMPLE is a convenient representations for optimizing the source code.

After GIMPLE, the source code is converted into the *static single assignment (SSA)* representation. The central idea of this form is the fact that each variable is assigned to only once, but can be used at the right hand side of an expression many times. Every time the same variable of a tree in the GIMPLE form is reassigned, the compiler creates a new version of that variable and stores the new value into it. When the same variable is assigned to in both branches of a conditional expression, one needs to merge the two possible values of the variable into a single variable. This operation is denoted as PHI function in the SSA form.

The SSA form is also used for optimizations. GCC performs more than 20 different optimizations on SSA trees. After the SSA optimization pass, the tree is converted back to the GIMPLE form which is then used to generate a *register-transfer language (RTL)* form of a tree. RTL is a hardware-based representation that corresponds to an abstract target architecture with an infinite number of registers. An RTL optimization pass optimizes the tree in the RTL form. Finally, a GCC back-end generates the assembly code for the target architecture using the RTL representation. Examples of back-ends are x86 back end, mips back end, etc.