GCC vs. LLVM

Gönül Trautmann, 02.05.2024

Übersicht

1. Was ist GCC?
2. Was ist LLVM?
3. GCC vs. LLVM
    a. Architektur
    b. Unterschiede und Gemeinsamkeiten

Was ist GCC?

- GCC ➜ heute: GNU Compiler Collection
- Sammlung von Compiler für C, C++, Objective-C, D, Fortran, Ada, Go
- Erste Veröffentlichung 1987 im Rahmen des GNU-Projekts
- Standard-Compiler auf Linux

Was ist LLVM?

- LLVM stand früher für Low Level Virtual Machine, ist aber heute der Markenname.
- Entwicklungsbeginn: 2000 im Rahmen einer Forschungsarbeit
- Nicht nur eine Sammlung von Compiler: LLVM ist zusätzlich eine Sammlung von Bibliotheken und Werkzeugen zur Entwicklung von Compiler und Virtuellen Maschinen

Was ist LLVM?

- Großer Fokus auf Modularität und Wiederverwendbarkeit der Komponenten.

- Kompilierungsprozess besteht aus unabhängigen Teilschritten und ist transparent

GCC vs. LLVM - Architektur

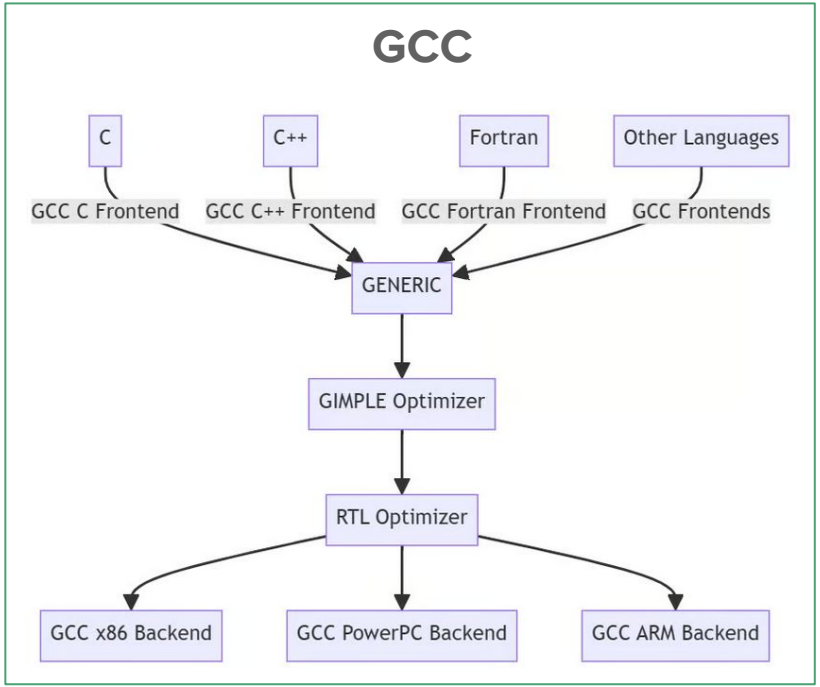


Abb. 1: GCC-Architektur

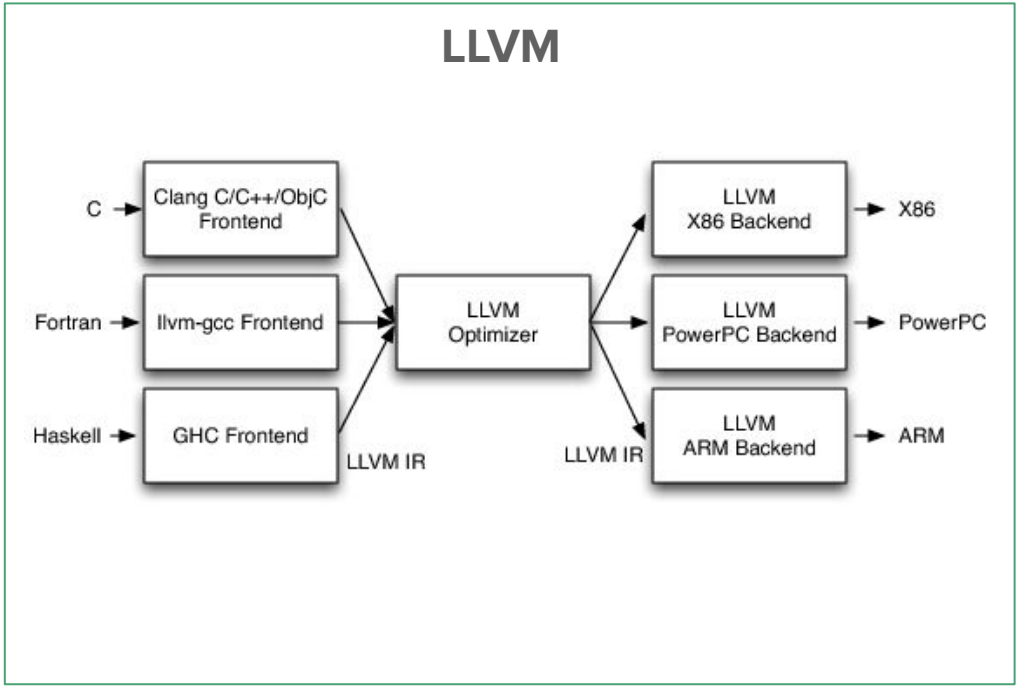


Abb. 2: LLVM-Architektur

GCC vs. LLVM - Unterschiede und Gemeinsamkeiten

	GCC	**LLVM**
Architektur	Monolithischer Compiler	Modulares Framework aus Bibliotheken, APIs, Debugger und anderen diversen Tools, die miteinander verkettet werden können
Open source	ja (restriktive Regeln)	ja (flexiblere Regeln)
Unterstützte Sprachen	C, C++, Objective-C, D, Fortran, Ada, Go	wie GCC, ~~Go~~, Swift, Java, Delphi, Julia, Haskell, Dylan, Gambas, Python, Ruby, Rust, ActionScript, Vala, Zig, Genie, GLSL

GCC vs. LLVM

	GCC	**LLVM**
Performance/ Benchmark	Keine eindeutige Aussage möglich, da je nach Anwendung, Workstation oder Prozessor mal GCC oder LLVM besser performed	
Unterstützte Betriebssysteme	POSIX kompatible Plattformen. Aufruf: Linux → gcc bzw. ggcrs für Rust Mac → homebrew Windows: → MinGW → Cygwin	Plattformunabhängig. Aufruf hängt von der Programmiersprache ab, z.B.: Rust → rustc C/C++ → clang Python → Nutzung von numba direkt im Code als just-in-time Compiler

Vielen Dank. Fragen?

Quellen

https://gcc.gnu.org/

https://llvm.org/

https://de.wikipedia.org/wiki/GNU_Compiler_Collection

https://de.wikipedia.org/wiki/LLVM

https://opensource.com/article/22/5/gnu-c-compiler

https://datascientest.com/de/llvm-alles-ueber-diesen-compiler

https://blog.logrocket.com/exploring-rust-compiler-options-gcc-vs-llvm/

https://chat.openai.com/

Benchmark: https://www.phoronix.com/review/gcc14-clang18-amd-zen4/5

Bildquellen
Abb 1: https://blog.logrocket.com/exploring-rust-compiler-options-gcc-vs-llvm/
Abb. 2: https://aosabook.org/en/v1/llvm.html

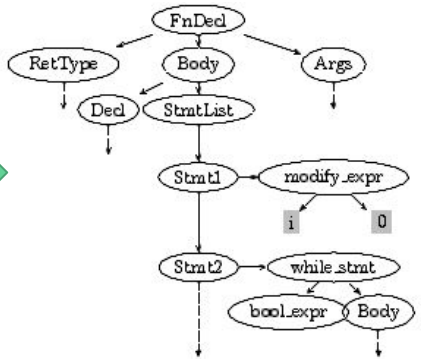
Backup: GCC-Codebeispiele beim Kompilierungsvorgang

Code/Programm

```
int f(char *a)
{
  int n = 10; int i, g;

  i = 0;
  while (i < n) {
    a[i] = g * i + 3;
    i = i + 1;
  }
  return i;
}
```

GENERIC → AST



GIMPLE

```
goto <D.1197>;
<D.1196>:;
a = a + 1;
<D.1197>:;
if (a <= 7)
    goto <D.1196>;
else
    goto <D.1198>;
<D.1198>:;
```

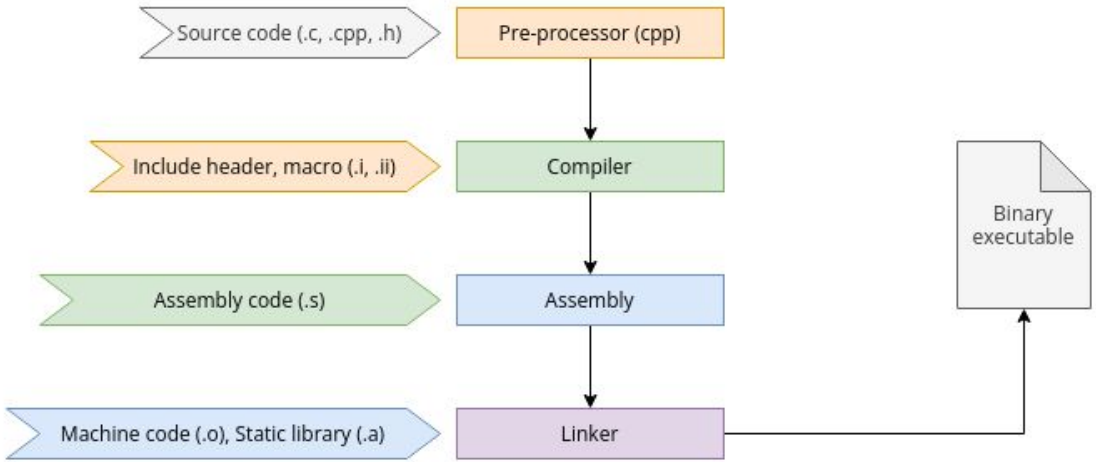
RTL

```
(set (reg:SI 140)
   (plus:SI (reg:SI 138)
            (reg:SI 139)))
```

Backup - GCC Aufruf und Struktur

```
$ gcc -v -o hellogcc hellogcc.c
```

gccrs für Rust



(Jayashree Huttanagoudar, CC BY-SA 4.0)

Präprozessor: Macht die Vorarbeit (parsen der #includes, #defines verarbeiten, Kommentare entfernen,..)

Compiler: Sprache wird erkannt und Assembly Code für entsprechende CPU wird erstellt.

Assembler konvertiert Code in Machinen-Sprache

Linker verlinkt den Machinen-Code mit der Library, um die ausführbare Datei zu erstellen