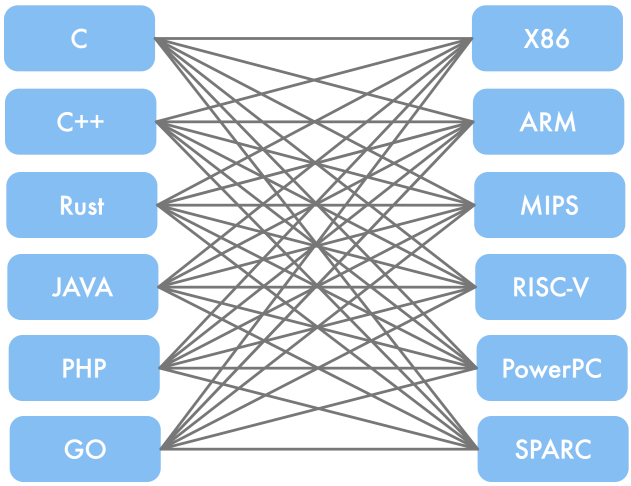
> 感谢up主 ZOMI酱：https://space.bilibili.com/517221395

GCC编译过程和原理

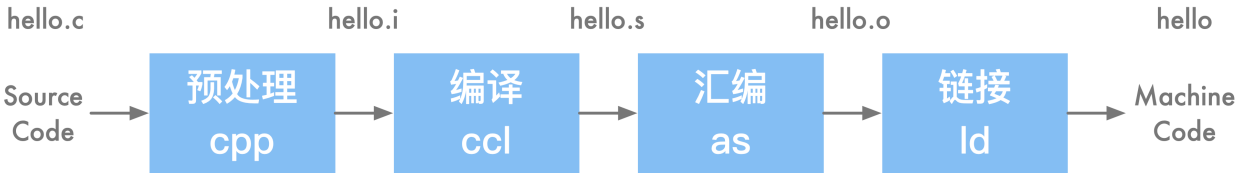
GCC的主要特征

- 是一个可移植的编译器，支持多种硬件平台
- 跨平台交叉编译
- 有多种语言前端，用于解析不同的语言
- 模块化设计，可加入新语言和新CPU架构支持
- 是开源自由软件，可免费使用



GCC的编译流程

GCC的编译过程可以大致分为预处理、编译、汇编和链接四个阶段。



源程序（文本）

```c
#include <stdio.h>

#define HELLOWORD ("hello world\n")

int main(void){
    printf(HELLOWORD);
    return 0;
}
```

预处理(cpp)

生成文件 hello.i

```
gcc -E hello.c -o hello.i
```

在预处理过程中，源代码会被读入，并检查其中包含的预处理指令和宏定义，然后进行相应的替换操作。此外，预处理过程还会删除程序中的注释和多余空白字符。最终生成的.i文件包含了经过预处理后的代码内容。

当高级语言代码经过预处理生成.i文件时，预处理过程会涉及宏替换、条件编译等操作。以下是对这些预处理操作的解释：

1. **头文件展开**：
   在预处理阶段，编译器会将源文件中包含的头文件内容插入到源文件中对应的位置，以便在编译时能够访问头文件中定义的函数、变量、宏等内容。
2. **宏替换**：
   在预处理阶段，编译器会将源文件中定义的宏在使用时进行替换，即将宏名称替换为其定义的内容。这样可以简化代码编写，提高代码的可读性和可维护性。
3. **条件编译**：
   通过预处理指令如#if、#else、#ifdef等，在编译前确定某些代码片段是否应被包含在最终的编译过程中。这样可以根据条件编译选择性地包含代码，实现不同平台、环境下的代码控制。
4. **删除注释**：
   在预处理阶段，编译器会删除源文件中的注释，包括单行注释（//）和多行注释（/.../），这样可以提高编译速度并减少编译后代码的大小。
5. **添加行号和文件名标识**：
   通过预处理指令如#line，在预处理阶段添加行号和文件名标识到源文件中，便于在编译过程中定位错误信息和调试。

6. **保留#pragma命令**：
   在预处理阶段，编译器会保留以#pragma开头的预处理指令，如#pragma once、#pragma pack等，这些指令可以用来指导编译器进行特定的处理，如控制编译器的行为或优化代码。

`hello.i` 文件部分内容如下，详细可见 `` `../code/gcc/hello.i `` 文件。

```
int main(void){
    printf(("hello world\n"));
    return 0;
}
```

在该文件中，已经将头文件包含进来，宏定义HELLOWORD替换为字符串"hello world\n"，并删除了注释和多余空白字符。

编译(ccl)

在这里，编译并不仅仅指将程序从源文件转换为二进制文件的整个过程，而是特指将经过预处理的文件（`hello.i`）转换为特定汇编代码文件（`hello.s`）的过程。

在这个过程中，经过预处理后的.i文件作为输入，通过编译器（ccl）生成相应的汇编代码.s文件。编译器（ccl）是GCC的前端，其主要功能是将经过预处理的代码转换为汇编代码。编译阶段会对预处理后的.i文件进行语法分析、词法分析以及各种优化，最终生成对应的汇编代码。

汇编代码是以文本形式存在的程序代码，接着经过编译生成.s文件，是连接程序员编写的高级语言代码与计算机硬件之间的桥梁。

生成文件 `hello.s`：

```
gcc -S hello.i -o hello.s
```

`hello.s`：

```
    .section    __TEXT,__text,regular,pure_instructions
    .build_version macos, 10, 15    sdk_version 10, 15, 6
    .globl    _main                 ## -- Begin function main
    .p2align    4, 0x90
_main:                              ## @main
    .cfi_startproc
## %bb.0:
    pushq    %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset %rbp, -16
    movq    %rsp, %rbp
```

```
    .cfi_def_cfa_register %rbp
    subq    $16, %rsp
    movl    $0, -4(%rbp)
    leaq    L_.str(%rip), %rdi
    movb    $0, %al
    callq    _printf
    xorl    %ecx, %ecx
    movl    %eax, -8(%rbp)          ## 4-byte Spill
    movl    %ecx, %eax
    addq    $16, %rsp
    popq    %rbp
    retq
    .cfi_endproc
                                        ## -- End function
    .section    __TEXT,__cstring,cstring_literals
L_.str:                                 ## @.str
    .asciz    "hello world\n"

.subsections_via_symbols
```

现在`hello.s`文件中包含了完全是汇编指令的内容，表明`hello.c`文件已经被成功编译成了汇编语言。

汇编(as)

在这一步中，我们将汇编代码转换成机器指令。这一步是通过汇编器(as)完成的。汇编器是GCC的后端，其主要功能是将汇编代码转换成机器指令。

汇编器的工作是将人类可读的汇编代码转换为机器指令或二进制码，生成一个可重定位的目标程序，通常以.o作为文件扩展名。这个目标文件包含了逐行转换后的机器码，以二进制形式存储。这种可重定位的目标程序为后续的链接和执行提供了基础，使得我们的汇编代码能够被计算机直接执行。

生成文件 `hello.o`

```
gcc -c hello.s -o hello.o
```

链接(ld)

链接过程中，链接器的作用是将目标文件与其他目标文件、库文件以及启动文件等进行链接，从而生成一个可执行文件。在链接的过程中，链接器会对符号进行解析、执行重定位、进行代码优化、确定空间布局，进行装载，并进行动态链接等操作。通过链接器的处理，将所有需要的依赖项打包成一个在特定平台可执行的目标程序，用户可以直接执行这个程序。

```
gcc -o hello.o -o hello
```

添加-v参数，可以查看详细的编译过程：

```
gcc -v hello.c -o hello
```

- **静态链接**是指**在链接程序时，需要使用的每个库函数的一份拷贝被加入到可执行文件中**。通过静态链接使用静态库进行链接，生成的程序包含程序运行所需要的全部库，可以直接运行。然而，静态链接生成的程序体积较大。
- **动态链接**是指**可执行文件只包含文件名，让载入器在运行时能够寻找程序所需的函数库**。通过动态链接使用动态链接库进行链接，生成的程序在执行时需要加载所需的动态库才能运行。相比静态链接，动态链接生成的程序体积较小，但是必须依赖所需的动态库，否则无法执行。

编译方法

类型	定义	示例
本地编译	编译源代码的平台和执行源代码编译后程序的平台是同一个平台。	在Intel x86架构/Windows平台上编译，生成的程序在同样的Intel x86架构/Windows 10下运行。
交叉编译	编译源代码的平台和执行源代码编译后程序的平台是两个不同的平台。	在Intel x86架构/Linux（Ubuntu）平台上使用交叉编译工具链编译，生成的程序在ARM架构/Linux下运行。

GCC 与传统编译过程区别

传统的三段式划分是指将编译过程分为前端、优化、后端三个阶段，每个阶段都有专门的工具负责。

而在GCC中编译过程被分成了预处理、编译、汇编、链接四个阶段 。其中 GCC 的预处理、编译阶段属于三段式划分的前端部分，汇编阶段属于三段式划分的后端部分。

GCC 的链接阶段是三段式划分后端部分的优化阶段合并，但其与端部分的目的一致，都是为了生成可执行文件。

GCC 编译过程的四个阶段与传统的三段式划分的前端、优化、后端三个阶段有一定的重合和对应关系，但GCC更为详细和全面地划分了编译过程，使得每个阶段的功能更加明确和独立。

总结

本节介绍了GCC的编译过程，主要包括预处理、编译、汇编和链接四个阶段。并总结了 GCC 的优点和缺点：

GCC 的优点	GCC 的缺点
1）支持 JAVA/ADA/FORTRAN	1）GCC 代码耦合度高，很难独立，如集成到专用 IDE 上，模块化方式来调用 GCC 难
2）GCC 支持更多平台	2）GCC 被构建成单一静态编译器，使得难以被作为 API 并集成到其他工具中
3）GCC 更流行，广泛使用，支持完备	3）从 1987 年发展到 2022 年 35 年，越是后期的版本，代码质量越差
4）GCC 基于 C,不需要 C++ 编译器即可编译	4）GCC 大约有 1500 万行代码，是现存大的自由程序之一