

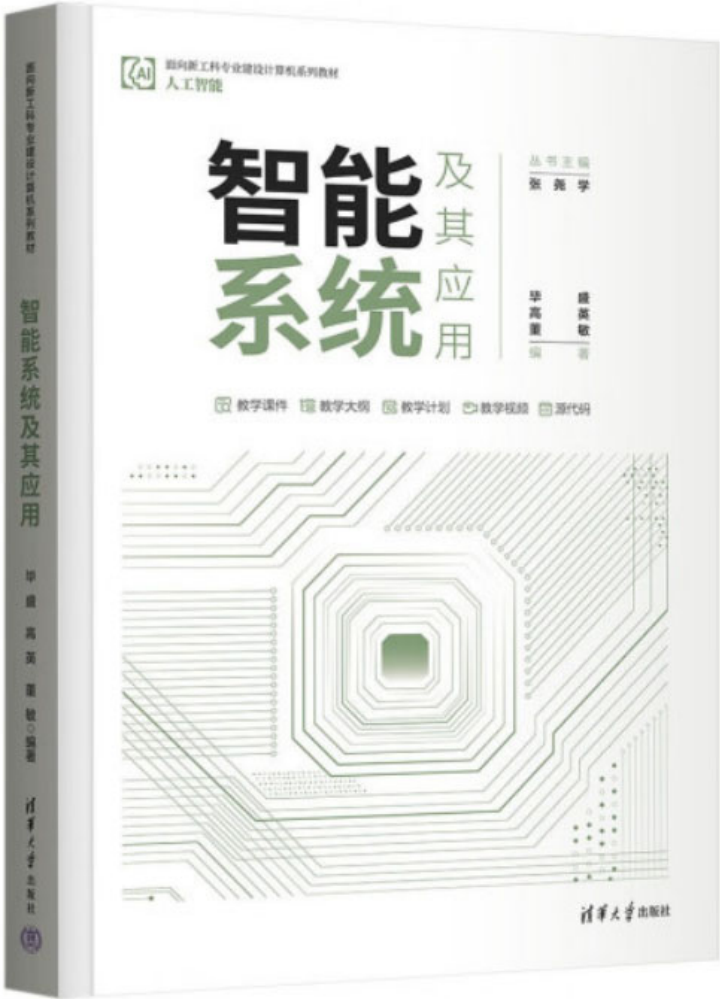


第3章 编译系统

智能系统及其应用
配套PPT



书名	书号	作者	出版社
智能系统及其应用	ISBN 978-7-302-60969-8	毕盛 高英 董敏	清华大学出版社





第3章 编译系统

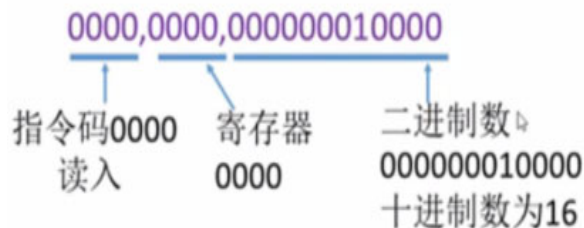
3.1 编译系统概述

3.2 编译器

3.3 常见编译器

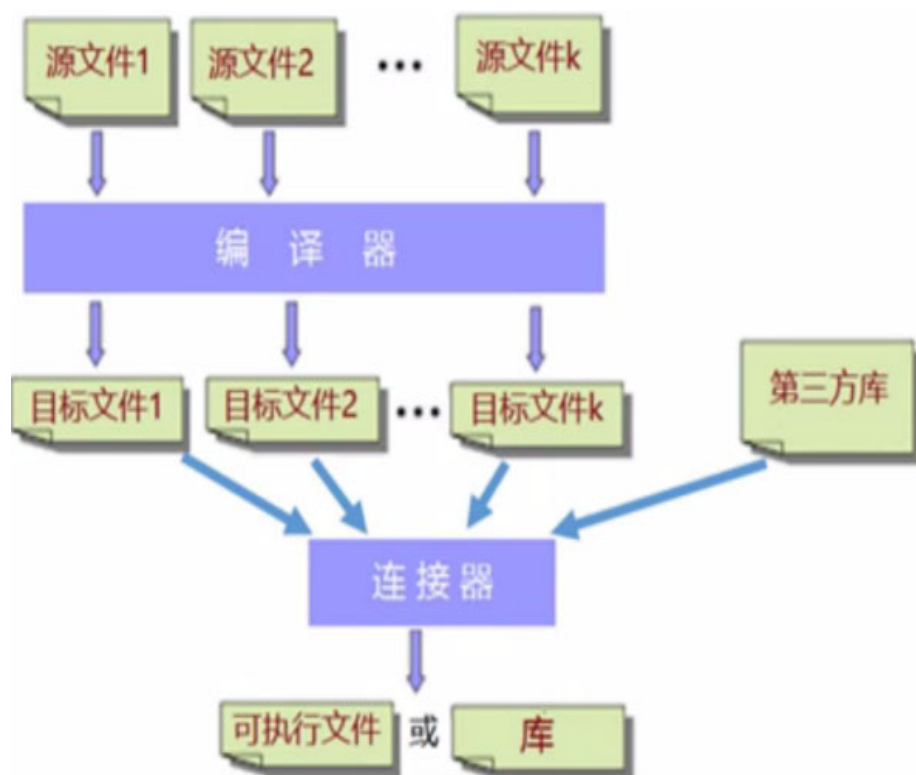
3.1 编译系统概述

机器语言是用二进制代码表示的计算机能直接识别和执行的一种机器指令集合：



汇编语言用字母和数字表示对应的0，1串指令及数据。如用 `MOV A,16` 表示 `0000,0000,000000010000`；汇编语言和机器语言一对一对应，比机器语言可读性好、编程效率高、调试性好。但汇编语言仍不具有移植性，且与人类语言差异很大。

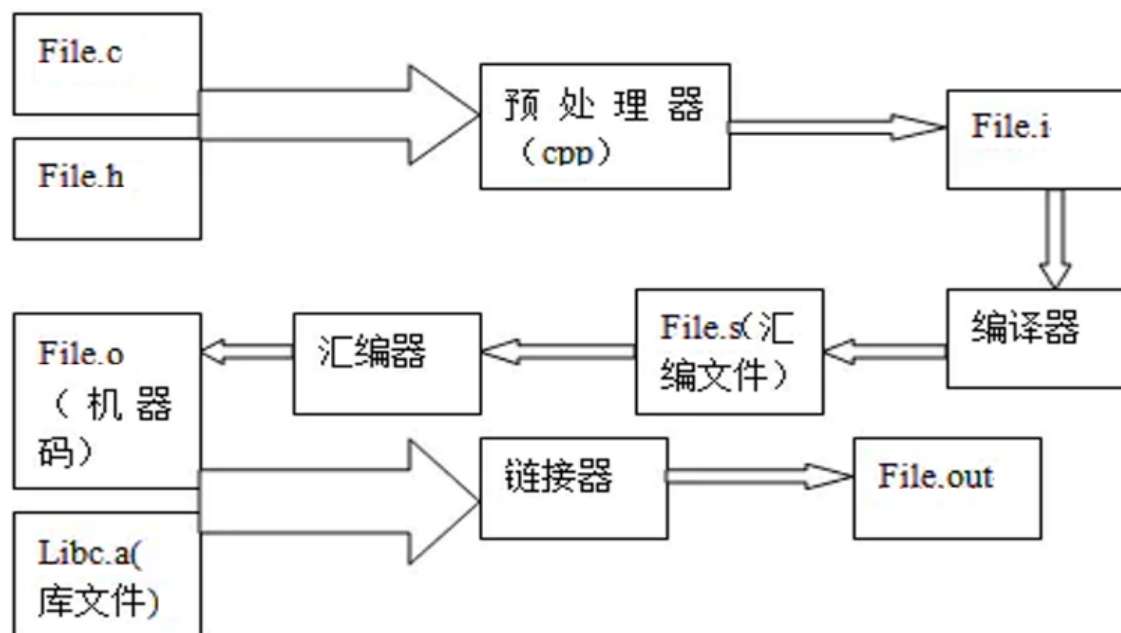
因此计算机公司提出了多种高级语言，从而便于开发人员开发程序，常见的开发语言有C、C++、python和java等。而如何把高级语言对应到汇编语言和机器语言这些低级语言，就需要有编译器来完成。



3.2 编译器

高级语言编译生成可以在芯片上运行机器二进制机器码的过程如下图所示。

3.2.1 编译器流程说明



传统的三段编译器设计中，前端负责解析源码，检查源码错误以及建立抽象语法树来生成编译中间件（IR）；优化器负责将中间件进行逻辑重组以及优化；后端负责按照代码运行环境生成机器码并进行链接优化，libc参与静态库链接以及，运行时的so库调用。



(1) 编译器 (compiler) : 将源语言翻译成目标语言。重要任务是在翻译过程中发现源语言中存在的错误。相比于解释器, 目标语言执行的速度快。

词法分析(Lexical Analysis)、语法分析 (Syntax Analysis)、
语义分析 (Semantic Analysis)、中间代码生成、优化和代码生成

(2) 解释器 (interpreter) : 利用用户的输入执行源程序中指定的操作。相比于编译器, 错误诊断效果好, 解释器是逐个语句执行。

Visual Basic、Java、JavaScript 都是解释执行的 (其中有些语言也可以编译执行)


(3) 预处理器 (preprocessor) : 把源程序聚合在一起, 把宏的缩写转换为源语言的语句。



(4) 汇编器 (**assembler**) : 将汇编语言程序处理后生成可重定位的机器代码。

(5) 链接器 (**linker**) : 解决外部内存地址问题。将多个可重定位的目标文件以及库文件链接到一起，形成真正在机器上运行的代码。

(6) 加载器 (**loader**) : 把所有的可执行目标文件放到内存中执行。



3.2.2 链接过程说明

链接过程是将多个可重定位的目标文件以及库文件链接到一起，形成真正在机器上运行的代码。

1. 目标文件

以Linux系统为例，目标文件如下几类：

(1) 可重定位文件：如.o文件，包含代码和数据，可以被链接成可执行文件或共享目标文件，静态链接库属于这一类。

(2) 可执行文件：如/bin/bash文件，包含了可以直接执行的程序，一般没有扩展名。

(3) 共享目标文件：如.so文件，包含代码和数据，可以跟其他可重定位文件和共享目标文件链接产生新的目标文件，也可以跟可执行文件结合作为进程映像的一部分。

(4) 目标文件由许多段组成，其中主要的段包括：


代码段 (.text)：保存编译后得到的指令数据。

数据段 (.data)：保存已经初始化的全局静态变量和局部静态变量。

只读数据段 (.rodata)：保存只读变量和字符串常量，有些编译器会把字符串常量放到”.data”段。

BSS段 (.bss)：保存未初始化的全局变量和局部静态变量。

重定位表：链接器在处理目标文件的时候，需要对目标文件中某些部位进行重定位，即代码段和数据段中那些绝对地址的引用位置，这些重定位信息记录在重定位表里。



2. 静态链接


几个目标文件进行链接时，每个目标文件都有其自身的代码段、数据段等，链接器需要将它们各个段的合并到输出文件中，具体有两种合并方法：

（1）按序叠加：将输入的目标文件按照次序叠加起来。这种方法会产生很多零散的段，而且每个段有一定的地址和空间对齐要求，会造成内存空间大量的内部碎片。所以这个方法现在较少用。

（2）第二种方法分为2个步骤进行：

1）空间与地址分配。扫描所有输入的目标文件，获得各个段的长度、属性和位置，收集它们符号表中所有的符号定义和符号引用，统一放到一个全局符号表中。此时，链接器可以获得所有输入目标文件的段长度，将他们合并，计算出输出文件中各个段合并后的长度与位置并建立映射关系。

2）符号解析与重定位。经过第一步后，输入文件中的各个段在链接后的虚拟地址已经确定了，链接器开始计算各个符号的虚拟地址。各个符号在段内的相对地址是固定的，链接器只需要给他们加上一个偏移量，调整到正确的虚拟地址即可。



3. 可执行文件的装载

可执行文件只有被装载到内存以后才能运行，最简单的办法是把所有的指令和数据全部装入内存，但这可能需要大量的内存，为了更有效地利用内存，根据程序运行的局部性原理，可以把程序中最常用的部分驻留内存，将不太常用的数据放在硬盘中，即动态装入。

现在大部分操作系统采用的是页映射的方法进行程序装载，将内存和所有硬盘中的数据和指令按页为单位划分成若干个页，以后所有的装载和操作的单位就是页。



4. 动态链接

动态链接可以解决空间浪费和更新困难的问题，程序运行时才对目标文件进行链接。使用了动态链接之后，系统会首先加载该程序依赖的其他的目标文件，如果其他目标文件还有依赖，系统会按照同样方法将它们全部加载到内存。当所需要的所有目标文件加载完毕之后，如果依赖关系满足，系统开始进行链接工作，包括符号解析及地址重定位等。完成之后，系统把控制权交回给原程序，程序开始运行。此时如果运行第二个程序，它依赖于一个已经加载过的目标文件，则系统不需要重新加载目标文件，而只要将它们连接起来即可。



3.3 常见编译器

3.2.1 GCC编译器介绍

GCC（**GNU Compiler Collection**，**GNU编译器套件**）是由**GNU**开发的编程语言译器。**GNU编译器套件**包括**C**、**C++**、**Objective-C**、**Fortran**、**Java**、**Ada**和**Go**语言前端，也包括了这些语言的库（如**libstdc++**，**libgcj**等）。

GCC 编译器是 **Linux** 系统下最常用的 **C/C++** 编译器，大部分 **Linux** 发行版中都会默认安装。



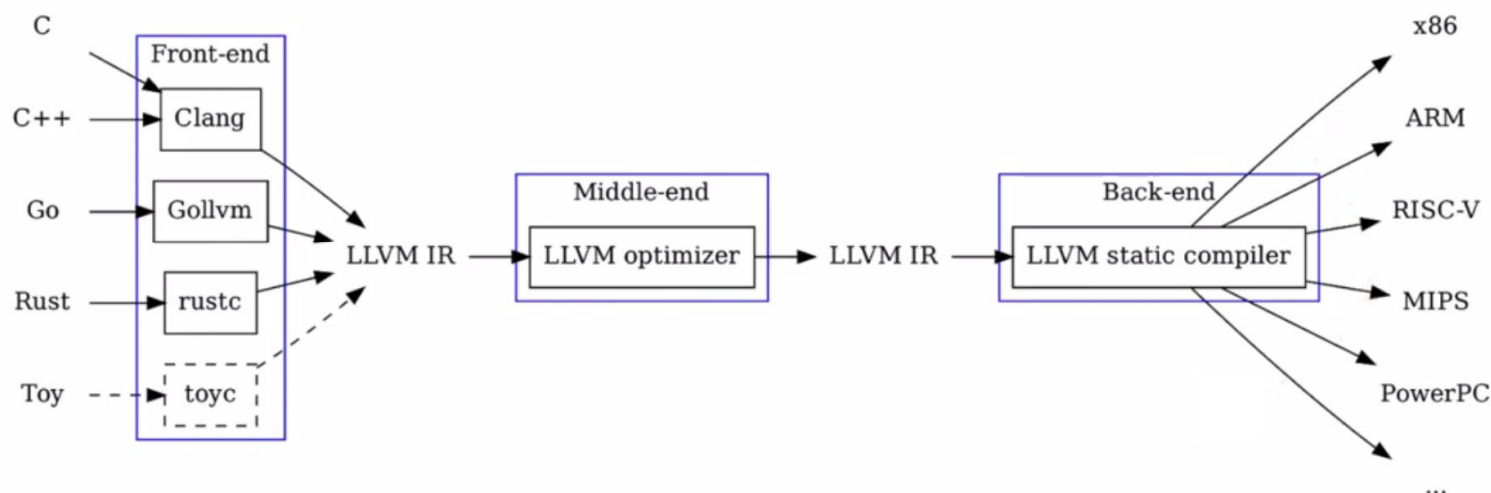
3.2.2 LLVM编译器介绍

LLVM (<http://llvm.org/>) 是构架编译器(compiler)的框架系统, 以C++编写而成, 用于优化以任意程序语言编写的程序的编译时间(compile-time)、链接时间(link-time)、运行时间(run-time)以及空闲时间(idle-time), 对开发者保持开放, 并兼容已有脚本。LLVM 核心库提供了与编译器相关的支持, 可以作为多种语言编译器的后台来使用。

前端：LLVM最初被用来取代GCC中的代码产生器，许多GCC的前端已经可以与其运行，LLVM目前支持Ada、C语言、C++、D语言、Fortran、Haskell、Julia、Objective-C、Rust及Swift的编译，它使用许多的编译器，有些来自4.0.1及4.2的GCC。

中间端：LLVM的核心是中间端表达式（Intermediate Representation, IR），一种类似汇编的底层语言。IR是一种强类型的精简指令集（Reduced Instruction Set Computing, RISC），并对目标指令集进行了抽象。例如，目标指令集的函数调用惯例被抽象为call和ret指令加上明确的参数。另外，IR采用无限个数的暂存器，使用如%0, %1等形式表达。LLVM支持三种表达形式：人类可读的汇编，在C++中对象形式和序列化后的bitcode形式。

后端：LLVM已经支持多种后端指令集，包括ARM、Qualcomm Hexagon、MIPS、Nvidia并行指令集（LLVM中称为NVPTX），PowerPC、AMD TeraScale、AMD GPU、SPARC、SystemZ、RISC-V、WebAssembly、x86、x86-64和XCore。





3.2.3 TVM (Tensor Virtual Machine)编译器

有关深度学习编译器框架，近年有名的是华盛顿大学陈天奇提出的TVM框架，它旨在缩小以生产力为中心的深度学习框架与以性能和效率为中心的硬件后端之间的差距。TVM与深度学习框架合作，为不同的后端提供端到端编译。TVM与LLVM的架构非常相似。TVM针对不同的深度学习框架和硬件平台，实现了统一的软件栈，以尽可能高效的方式，将不同框架下的深度学习模型部署到硬件平台上。

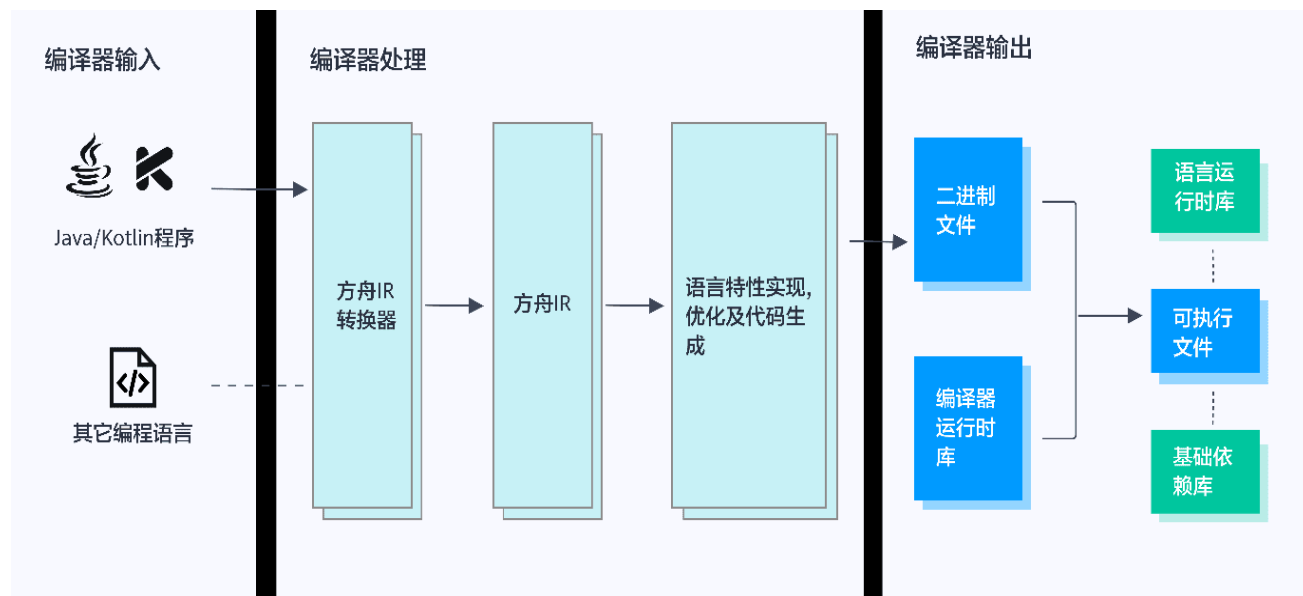
（1）构建了一个端到端的编译优化堆栈，允许将高级框架（如Caffe、MXNet、PyTorch、Caffe2、CNTK）专用的深度学习工作负载部署到多种硬件后端上（包括 CPU、GPU 和基于FPGA的加速器）。


（2）提供深度学习工作负载在不同硬件后端中的性能可移植性的主要优化挑战，并引入新型调度基元（**schedule primitive**）以利用跨线程内存重用、新型硬件内部函数和延迟隐藏。

（3）在基于FPGA的通用加速器上对TVM进行评估，以提供关于如何最优适应专用加速器的具体案例。

3.2.4方舟编译器

方舟编译器改变了系统及应用的编译和运行机制，直接将高级语言编译成机器码，让手机能直接听懂“高级语言”，消除了虚拟机动态编译的额外开销，提升了手机运行效率。同时，方舟编译器还能够理解程序特征、使用适合的指令来执行程序，因此能够极大程度地发挥出芯片的能力。目前，方舟编译器聚焦在 **Java** 代码性能上，未来，方舟编译器将覆盖多种编程语言（包括 **C/C++**、**JS** 等），多种芯片架构（包括**CPU**、**GPU**、**IPU**等），覆盖更广的业务场景。



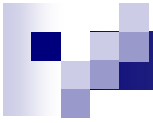


3.2.5 毕昇编译器

毕昇编译器是华为编译器实验室针对鲲鹏等通用处理器架构场景，打造的一款高性能、高可信及易扩展的编译器工具链，增强和引入了多种编译优化技术，支持C/C++/Fortran等编程语言。

毕昇编译器基于开源LLVM开发，并进行了优化和改进，LLVM是一种涵盖多种编程语言和目标处理器的编译器，毕昇编译器聚焦于对C、C++、Fortran语言的支持，利用LLVM的Clang作为C和C++的编译和驱动程序，Flang作为Fortran语言的编译和驱动程序。

毕昇编译器的运行平台是鲲鹏920硬件平台，支持的操作系统有openEuler21.03、openEuler 20.03 (LTS)、CentOS 7.6、Ubuntu 18.04、Ubuntu 20、麒麟V10和UOS 20。



谢谢！