[第1章 编译概述 3](#_Toc425695288)

[1.1 编译流程 3](#_Toc425695289)

[1.1.1编译器简介 3](#_Toc425695290)

[1.1.2前端 5](#_Toc425695291)

[1.1.3 优化器 5](#_Toc425695292)

[1.1.4 后端 5](#_Toc425695293)

[1.2 平台简介 5](#_Toc425695294)

[1.2.1 基于MIPS的硬件平台简介 5](#_Toc425695295)

[1.2.2 MIPS架构简介 6](#_Toc425695296)

[1.3 数据结构 6](#_Toc425695297)

[1.3.1 语法树 6](#_Toc425695298)

[1.3.2 符号表 6](#_Toc425695299)

[第2章 词法分析 6](#_Toc425695300)

[2.1 正则表达式 6](#_Toc425695301)

[2.2 有穷自动机 6](#_Toc425695302)

[2.2.1 确定型有穷自动机 6](#_Toc425695303)

[2.2.2 非确定型有穷自动机 6](#_Toc425695304)

[2.3 词法分析器设计 6](#_Toc425695305)

[2.3.1 lex简介 6](#_Toc425695306)

[2.3.2 lex语法 6](#_Toc425695307)

[2.3.3 使用flex生成语法分析器 6](#_Toc425695308)

[第3章 语法分析 6](#_Toc425695309)

[3.1 上下文无关语法 6](#_Toc425695310)

[3.1.1 正则表达式的局限性 7](#_Toc425695311)

[3.1.2 上下文无关语法 7](#_Toc425695312)

[3.1.3 语法分析树 7](#_Toc425695313)

[3.1.3 二义性语法 7](#_Toc425695314)

[3.2 语法分析过程 7](#_Toc425695315)

[3.2.1 自顶向下语法分析 7](#_Toc425695316)

[3.2.2 自底向上语法分析 7](#_Toc425695317)

[3.3 语法分析器设计 7](#_Toc425695318)

[3.3.1 yacc简介 7](#_Toc425695319)

[3.3.2 yacc语法 7](#_Toc425695320)

[3.3.3 使用yacc生成语法分析器 7](#_Toc425695321)

[3.4 错误处理 7](#_Toc425695322)

[第4章 语义分析 7](#_Toc425695323)

[4.1 符号表 7](#_Toc425695324)

[4.1.1 符号表定义 7](#_Toc425695325)

[4.1.3 建立符号表 8](#_Toc425695326)

[4.1.3 使用符号表处理作用域 8](#_Toc425695327)

[4.2 类型检查 8](#_Toc425695328)

[4.2.1 声明的类型检查 8](#_Toc425695329)

[4.2.2 表达式的类型检查 8](#_Toc425695330)

[4.3 语义分析器设计 8](#_Toc425695331)

[第5章 内存布局 8](#_Toc425695332)

[5.1 地址空间 8](#_Toc425695333)

[5.1.1 栈和栈指针 8](#_Toc425695334)

[5.1.2 全局数据和GP指针 8](#_Toc425695335)

[5.2 栈帧 8](#_Toc425695336)

[5.2.1 fp寄存器 8](#_Toc425695337)

[5.2.2 栈帧内布局 8](#_Toc425695338)

[5.2.3 函数调用过程 8](#_Toc425695339)

[第6章 中间代码 9](#_Toc425695340)

[6.1 中间表示树 9](#_Toc425695341)

[6.2 中间表示树翻译 9](#_Toc425695342)

[第7章 代码生成 10](#_Toc425695343)

[7.1 MIPS指令集简介 10](#_Toc425695344)

[7.1.1 MIPS汇编 10](#_Toc425695345)

[7.1.2 MIPS流水线简介 10](#_Toc425695346)

[7.2 生成MIPS汇编 10](#_Toc425695347)

[7.3 机器码生成 10](#_Toc425695348)

[第8章 代码优化 10](#_Toc425695349)

[8.1 指令选择与调度 10](#_Toc425695350)

[8.2 寄存器分配 10](#_Toc425695351)

[8.3 循环优化 10](#_Toc425695352)

# 编译概述

你有想过你写的C语言程序为什么能在计算机上运行吗？我们知道处理器是无法识别C语言的，处理器只能识别机器码。显然运行C语言编写的程序，需要经过一个将代码转换为机器码的过程，这个过程就是编译。

编译是现代计算机体系中一个重要环节，它在降低了程序的开发难度的同时，大大缩短了程序的开发时间。本章将讲解编译器的基本结构以及编译器的工作流程。

## 编译器及其基本工作流程

### 编译器

编译器是将高级语言翻译为低级语言的程序。高级语言（High-level programming language）是一种高度封装过的编程语言。特点是易于编写和理解，具有较高的可读性。与之相对的低级语言或者说机器语言（machine language）是一种指令集，是一种面向机器的语言。

当然很多人在第一次接触编程时可能并没有接触过编译器。比如现在流行的语言python就不需要编译器，取而代之的是解释器。这种语言成为解释性语言。解释型语言不需要编译，程序在运行时才翻译成机器语言，每执行一次都要翻译一次。

相对地，另一种语言称之为编译型语言，需要通过编译器将其翻译为机器语言才可以执行。

### 编译流程



图 1‑1

典型的编译流程如图1-1所示。下面将解释各个步骤的工作。

预编译。预编译是一个将代码进行文本替换的预处理过程。严格意义上来说预编译不属于编译的范畴，在本书中我们不讨论预编译过程。

词法分析。**词法分析**（英语：**lexical analysis**）是计算机科学中将字符序列转换为单词（Token）序列的过程。进行词法分析的程序或者函数叫作**词法分析器**（Lexical analyzer，简称Lexer），也叫**扫描器**（Scanner）。词法分析器一般以函数的形式存在，供语法分析器调用。 完成词法分析任务的程序称为词法分析程序或词法分析器或扫描器。词法分析阶段是编译过程的第一个阶段，是编译的基础。这个阶段的任务是从左到右一个字符一个字符地读入源程序，即对构成源程序的字符流进行扫描然后根据构词规则识别单词(也称单词符号或符号)。词法分析程序实现这个任务。词法分析程序可以使用Lex等工具自动生成。

语法分析。**语法分析**（Syntactic analysis，也叫Parsing）是根据某种给定的形式文法对由单词序列（如英语单词序列）构成的输入文本进行分析并确定其语法结构的一种过程。[1]

**语法分析器**（Parser）通常是作为编译器或解释器的组件出现的，它的作用是进行语法检查、并构建由输入的单词组成的数据结构（一般是语法分析树、抽象语法树等层次化的数据结构）。语法分析器通常使用一个独立的词法分析器从输入字符流中分离出一个个的“单词”，并将单词流作为其输入。实际开发中，语法分析器可以手工编写，也可以使用工具（半）自动生成。

语义分析。语义分析是编译过程的一个逻辑阶段， 语义分析的任务是对结构上正确的源程序进行上下文有关性质的审查，进行类型审查。语义分析是审查源程序有无语义错误，为代码生成阶段收集类型信息。比如语义分析的一个工作是进行类型审查，审查每个算符是否具有语言规范允许的运算对象，当不符合语言规范时，编译程序应报告错误。如有的编译程序要对浮点数用作数组下标的情况报告错误。

中间代码生成。在进行了语法分析和语义分析阶段的工作之后，有的编译程序将[源程序](http://baike.baidu.com/view/546605.htm)变成一种内部表示形式，这种内部表示形式叫做中间语言或中间表示或中间代码。所谓“中间代码”是一种结构简单、含义明确的记号系统，这种记号系统复杂性介于源程序语言和机器语言之间，容易将它翻译成目标代码。另外，还可以在中间代码一级进行与机器无关的优化。产生中间代码的过程叫中间代码生成。

代码优化。所谓代码优化是指对程序代码进行等价（指不改变程序的运行结果）变换。程序代码可以是中间代码（如四元式代码），也可以是目标代码。等价的含义是使得变换后的代码运行结果与变换前代码运行结果相同。优化的含义是最终生成的目标代码短（运行时间更短、占用空间更小），时空效率优化。原则上，优化可以在编译的各个阶段进行，但最主要的一类是对中间代码进行优化，这类优化不依赖于具体的计算机。

目标代码生成。目标代码生成是编译的最后一个阶段。目标代码生成器把语法分析后或优化后的中间代码变换成目标代码。目标代码（object code）指计算机科学中编译器或汇编器处理[源代码](http://baike.baidu.com/view/60376.htm)后所生成的代码，它一般由机器代码或接近于机器语言的代码组成。本书生成的目标代码为汇编代码。

汇编。汇编大多是指汇编语言，汇编程序。把汇编语言翻译成机器语言的过程称为汇编。在汇编语言中，用助记符(Memoni)代替操作码，用地址符号(Symbol)或标号(Label)代替地址码。这样用符号代替机器语言的二进制码，就把机器语言变成了汇编语言。于是汇编语言亦称为符号语言。用汇编语言编写的程序，机器不能直接识别，要由一种程序将汇编语言翻译成机器语言，这种起翻译作用的程序叫汇编程序，汇编程序是系统软件中语言处理的系统软件。

### 编译器结构

上一节讲到了编译器的整个流程。

## 平台简介

### 1.2.1 基于MIPS的硬件平台简介



### MIPS架构简介

## 数据结构

### 语法树

### 符号表

# 词法分析

## 正则表达式

## 有穷自动机

### 确定型有穷自动机

### 非确定型有穷自动机

## 词法分析器设计

### lex简介

### lex语法

### 使用flex生成词法分析器

# 语法分析

## 上下文无关语法

### 正则表达式的局限性

### 上下文无关语法

### 语法分析树

### 二义性语法

## 语法分析过程

### 自顶向下语法分析

### 自底向上语法分析

## 语法分析器设计

### yacc简介

### yacc语法

### 使用yacc生成语法分析器

## 错误处理

# 语义分析

语义分析又称为上下文相关分析，是在词法分析和语法分析之后进一步检查语法错误的手段。即使是语法分析完全通过的程序，仍然会存在严重的程序错误，典型的如在C语言中使用未定义的变量的语句可以通过语法分析，但是显然这是不符合C语言规范的。而语法分析就是要将语句和上下文进行关联分析，从而能够发现程序在类型以及约定等方面的错误。

## 符号表

### 符号表定义

### 建立符号表

### 使用符号表处理作用域

## 类型检查

### 声明的类型检查

### 表达式的类型检查

## 语义分析器设计

# 中间代码生成

## 中间表示树

## 中间表示树翻译

# 代码优化

# 代码生成

## MIPS指令集简介

### MIPS汇编

### MIPS流水线简介

## 指令选择

## 指令调度

## 寄存器分配