山东大学
SHANDONG UNIVERSITY

编译原理

第一章 编译器概述

授 课 教 师 ：郑艳伟
手 机 ：18614002860（微信同号）
邮 箱 ：zhengyw@sdu.edu.cn

- 《编译原理与技术》研究将高级程序设计语言转化为可执行代码的原理和实现技术，是计算机专业最难的科目：
  - 编译原理使用数学工具研究语言问题；
  - 编译原理是一门算法密集型课程；
  - 与操作系统一样，编译器是连接硬件与软件的桥梁；
  - 不同于操作系统，编译器使用生成的代码操作硬件资源。
- 考核方式：考试卷面50% + 平时成绩50%
- 平时：作业 + 实验
- 教材和参考书
  - 许畅, 冯洋, 郑艳伟, 陈鄞. 编译方法、技术与实践[M]. 北京: 机械工业出版社, 2024.
  - 陈火旺等. 编译原理, 国防工业出版社第3版, ISBN: 978-7-118-02207-0.

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.1.1 语言的分类

- 编译原理的研究对象
  - ✓ 形式语言：是用精确的数学或机器可处理的公式定义的语言
    - 编译器的研究对象是程序设计语言，属于形式语言的一种，主要处理方法是符号主义
  - × 自然语言：就是人类讲的语言，比如汉语、英语、法语、德语等
    - 早期符号主义，现在连接主义

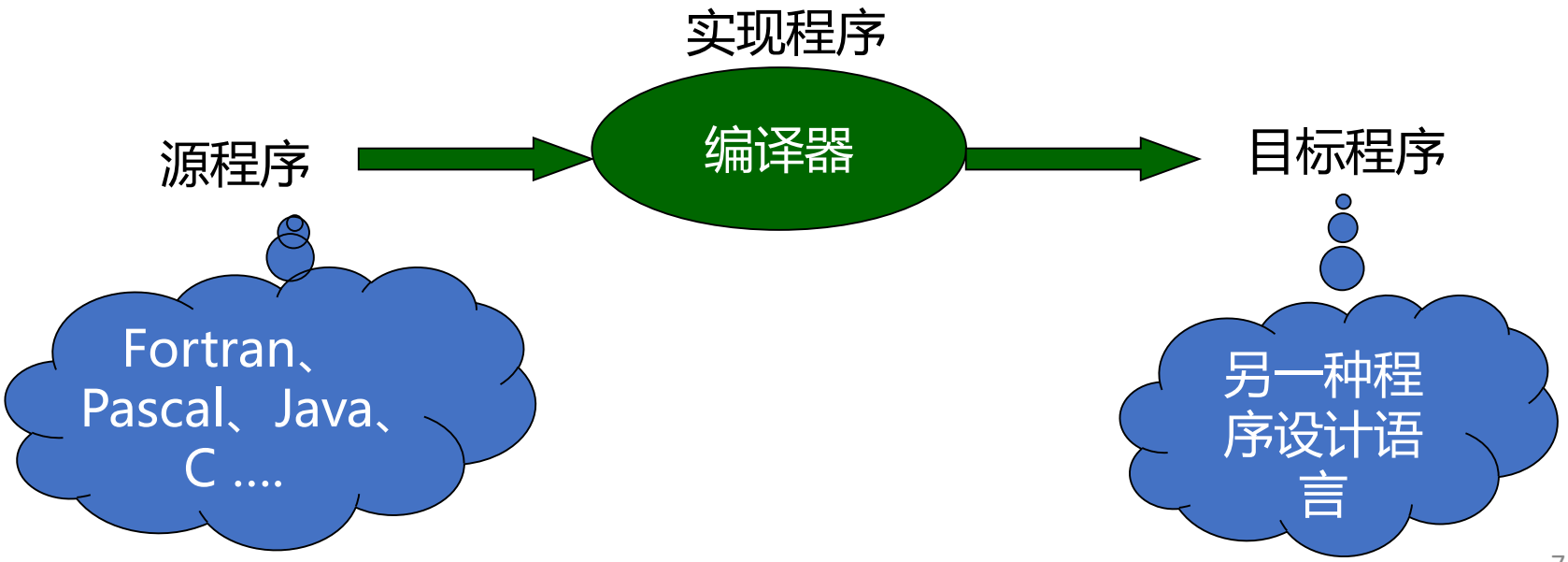
1.1.2 程序设计语言分类

- 机器语言：是机器能直接识别的程序语言或指令代码，主要对硬件进行操作。
- 汇编语言：将计算机指令用易于记忆的符号表示。
- 高级语言：由表达各种不同意义的“关键词”和“表达式”，按一定的语义规则组成的程序。
- 中间语言：介于高级语言和机器语言之间，既与硬件无关从而便于高级语言转换，又容易转换为机器语言或汇编语言。
- MSIL、Java ByteCode和.pyc ByteCode：属于在虚拟机上运行的中间语言，作用类似于汇编语言，但结构要比汇编语言高级的多，需要虚拟机二次编译或者解释才能执行。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.1.3 编译程序

- 翻译程序：把一种语言程序（源语言）转换成另一种语言程序（目标语言）
- 编译程序(Compiler)：源语言为高级语言，目标语言为低级语言的翻译程序。
- 解释程序(Interpreter)：以源语言作为输入，但不产生目标程序，而是边翻译边执行源程序本身。
- 汇编程序(Assembler)：是将汇编语言翻译为机器语言的程序，又称为汇编器。

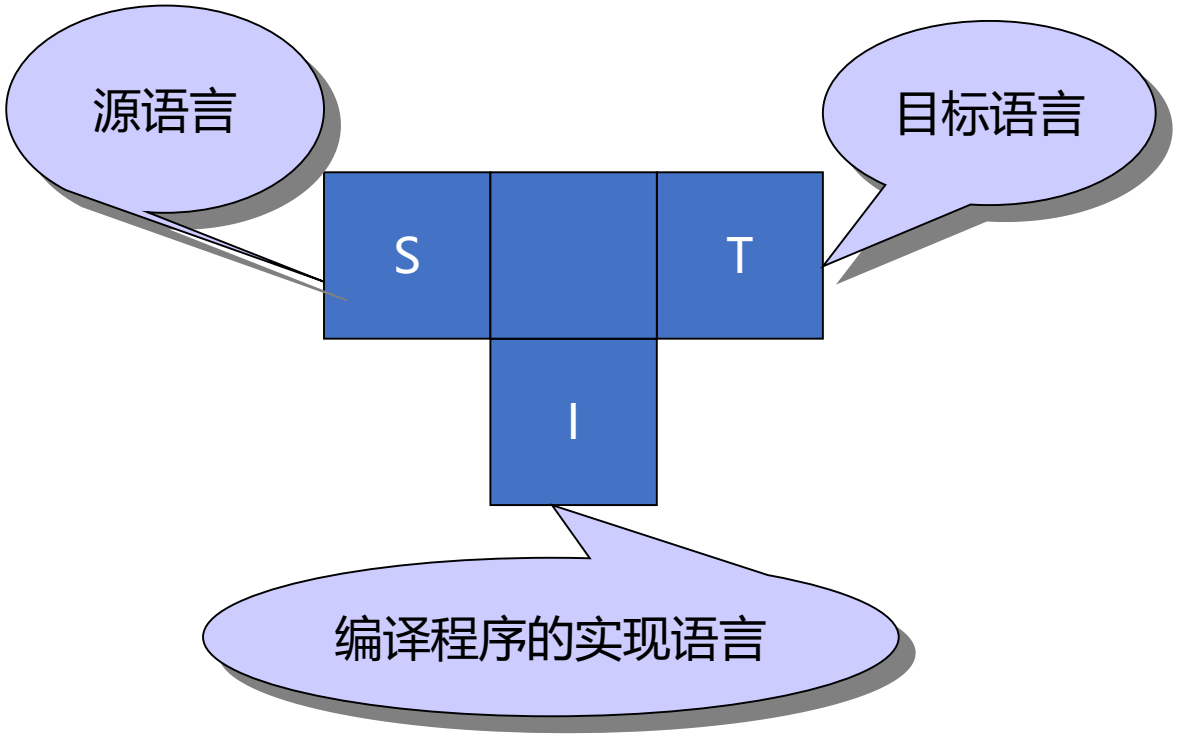


山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.1.3 编译程序

T形图

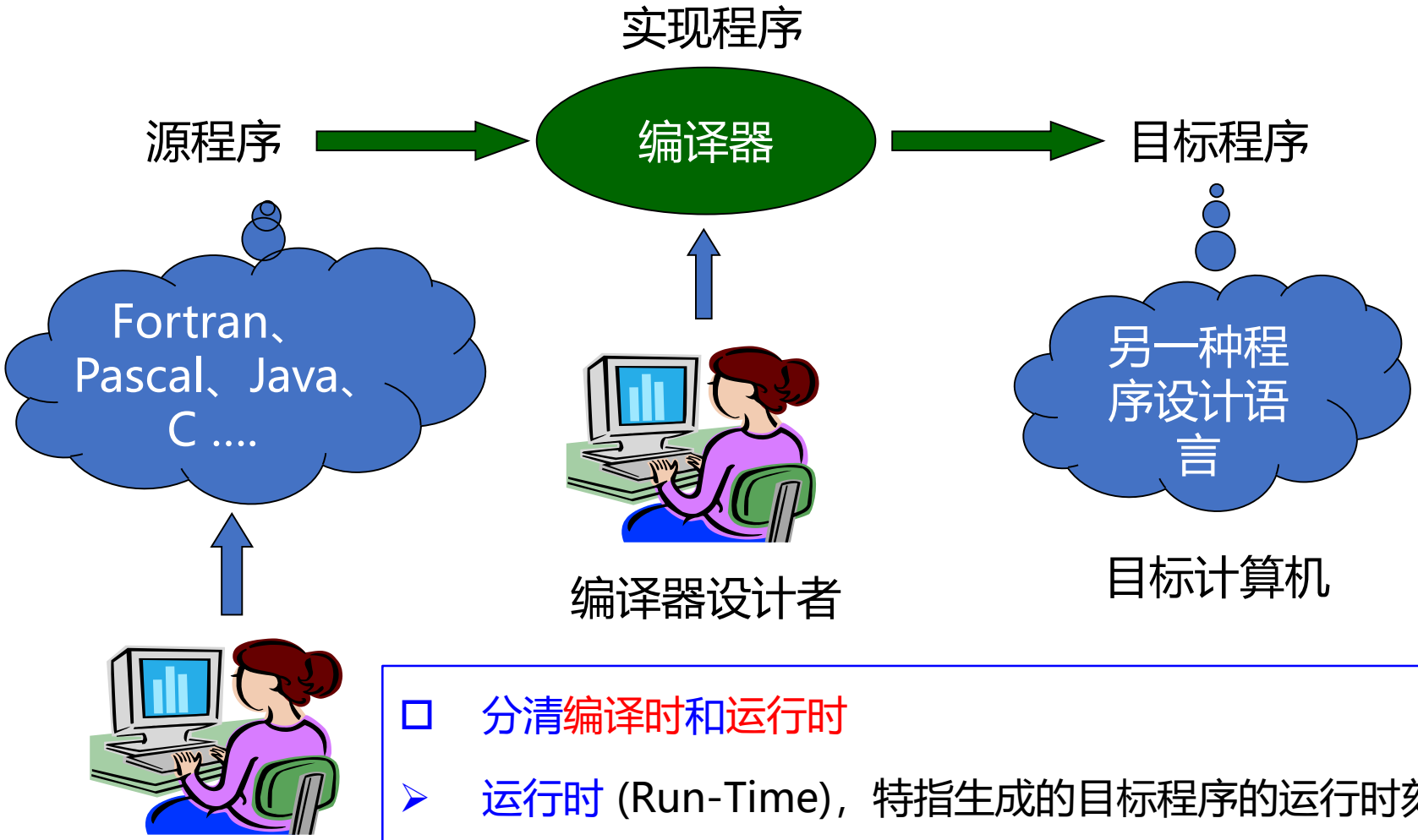
- 源语言S（Source）
- 目标语言T（Target）
- 编译程序实现语言I（Implementation）



1.1.4 编译原理与技术的特点

- 《编译原理与技术》研究将高级程序设计语言转化为可执行代码的原理和实现技术，是计算机专业最难的科目之一
  - 编译原理使用数学工具研究语言问题；
  - 编译原理是一门算法密集型课程；
  - 与操作系统一样，编译器是连接硬件与软件的桥梁；
  - 不同于操作系统，编译器使用生成的代码操作硬件资源。

1.1.4 编译原理与技术的特点



- 分清编译时和运行时
  - 运行时 (Run-Time)，特指生成的目标程序的运行时刻
  - 编译时 (Compile-Time)，指编译器的运行时刻

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.1.5 编译程序的生成

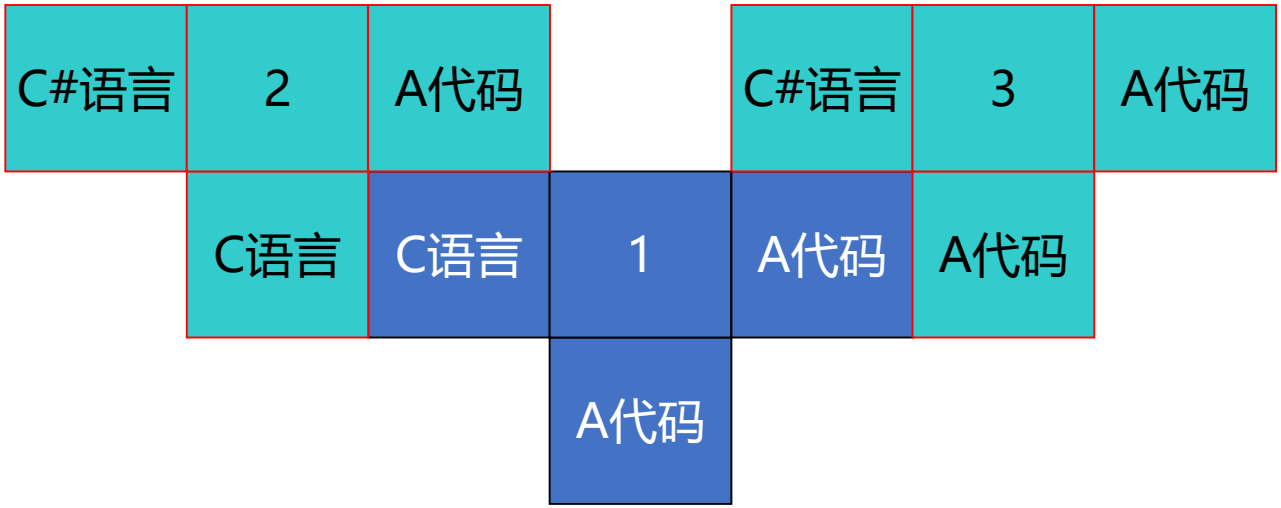
- 对机器的名称约定
  - 宿主机：运行编译程序的计算机。
  - 目标机：运行目标代码的计算机。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.1.5.1 设计新语言

- A机器上已有用A机器代码实现的高级语言C的编译程序

  → A机器上用A机器代码实现的高级语言C#的编译程序

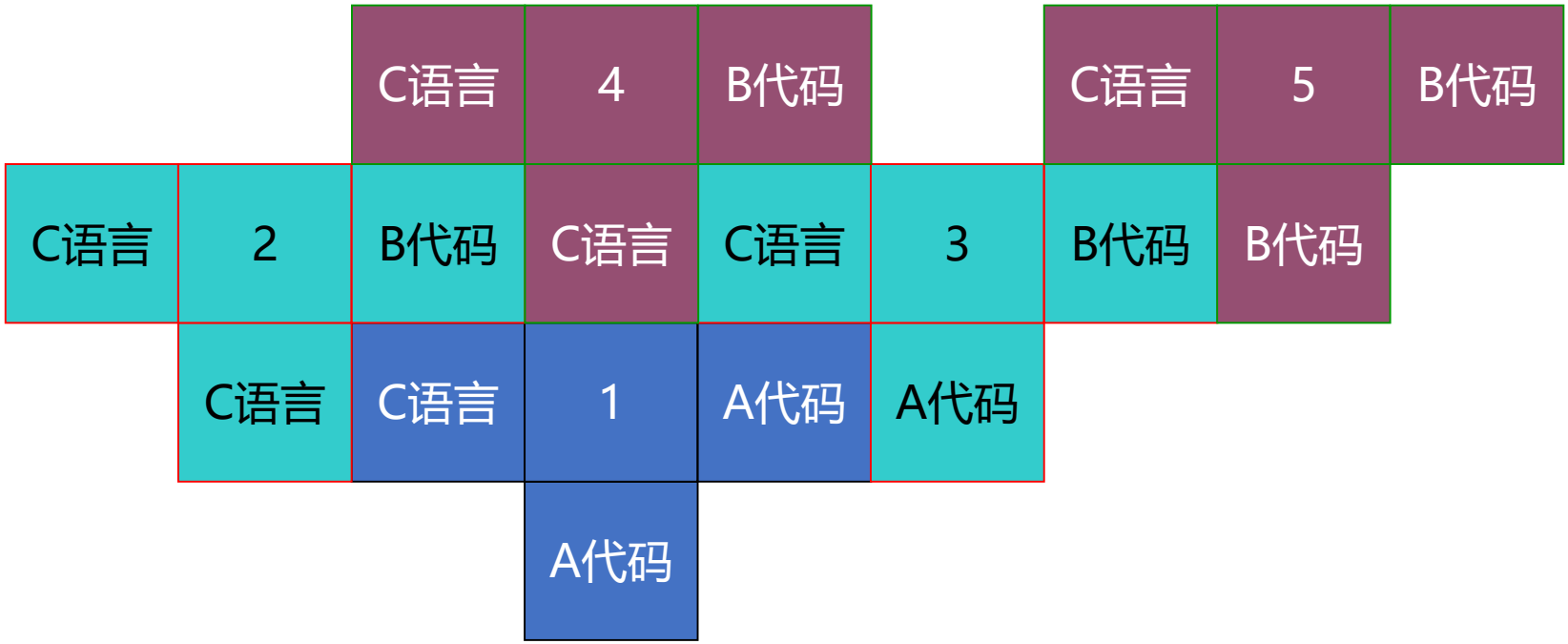


山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.1.5.2 移植到新机器

- A机器上已有用A机器代码实现的高级语言C的编译程序

  → B机器上用B机器代码实现的高级语言C的编译程序



山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.1.5.3 自编译生成编译器

- 自编译方式产生编译程序（自举）
  - 先对语言的核心部分构造一个小的编译程序；
  - 以它为工具构造一个能够编译更多语言成分的较大编译程序；
  - 如此扩展，最后形成所期望的整个编译程序。

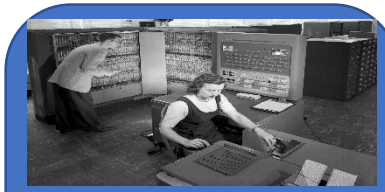
山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

高级程序设计语言的发展

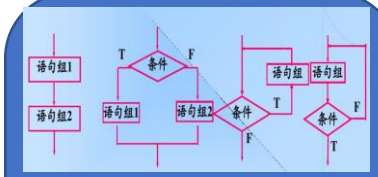
1801年，法国人Joseph Jacquard发明使用打孔卡片，程序和数据可以写到卡片上。

1946年2月14日，美国宾夕法尼亚大学研制出第一台通用计算机ENIAC，用插线板表示程序。



1956年，IBM推出704计算机，其上的符号汇编程序(SAP)是汇编发展中的一个重要里程碑。

针对汇编语言的缺点，IBM于1954年发布高级程序设计语言Fortran，1956年正式使用。



20世纪60~70年代：结构化设计方法，如Pascal、C等；后用软件工程方法开发更大规模程序。

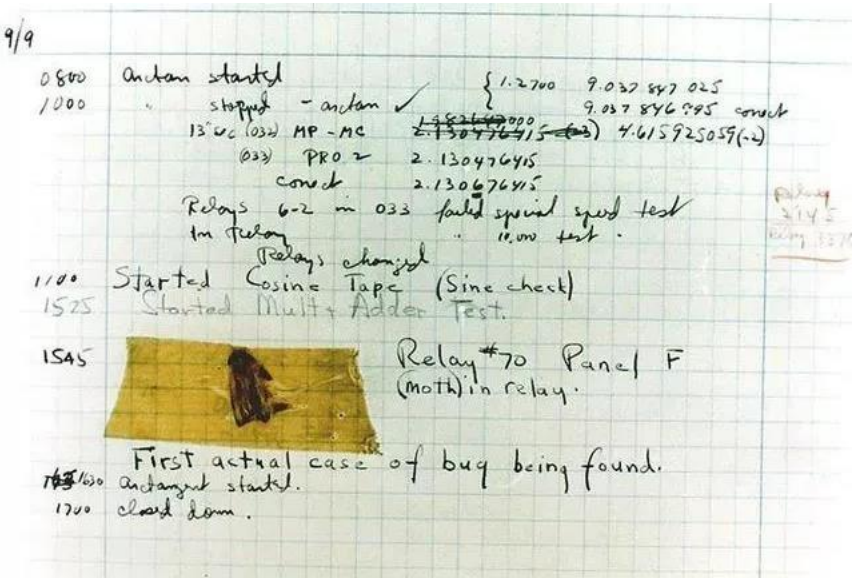
20世纪80年代：面向对象的程序设计语言Smalltalk问世，OOP立意于模拟现实世界。

20世纪90年代中期：基于可视化和面向对象的编程语言，如VB、VC++、Delphi等。

进一步发展：二进制级别软件复用、软件标准件的生产、组态平台（低代码平台）等。

Bug和Debug

- 1945年9月9日，Grace Hopper对Harvard Mark II设置好的17000个继电器进行编程后，正在进行整机运行，它突然停止了工作。
- 工作人员爬上去找原因，发现这台巨大的计算机内部一组继电器的触点之间有一只飞蛾（Bug），“卡”住了机器运行。
- Grace Hopper把这只飞蛾贴在管理日志上，写道：“就是这个bug，害得我们今天的工作无法完成”。
- 后来，实验室里的人每逢老板询问为何还没有做出结果，就把过错推给bug。
- 老板自然让赶快debug（除虫），因此bug和debug成为特指莫名其妙的“错误”和“排除错误”的专用词汇。



山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

高级语言分类

- 命令式编程(Imperative programming)，又称强制式语言、指令式语言，即利用命令式语言进行编程的方式，是一种描述计算机所需作出的行为的编程范型。命令式编程语言使用变量和更复杂的语句，但依从相同的范型，如C，C++, C# 和 Java。
- 声明式编程(Declarative programming)，即利用声明式语言进行编程的方式，与命令式编程相对立，它描述目标性质，让计算机明白目标，而非流程。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

命令式语言

- （1）面向过程的语言（Procedure-Oriented Language）
  - 以过程作为基本单元，每个过程由一系列语句组成，如C、Pascal、Fortran 等。
  - 该语言要求程序员告诉计算机每一步如何执行，最终组成一个完整的程序。

```
语句1;
语句2;
……
语句n;
```

```
1  int factorial(int n)
2  {
3    int f = 1;
4    for (; n > 0; n--)
5      f *= n;
6    return f;
7  }
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

命令式语言

- （2）面向对象的语言（Object-Oriented Language）
  - 是一类以对象作为基本程序结构单位的程序设计语言，其核心是将世界看做由对象组成，对象由数据和对数据的操作组成，而对象是程序运行时刻的基本成分。
  - 封装性，指把客观事物抽象为类，把类内部的实现隐藏，只向外公开接口进行操作，不支持对数据的直接操作，以保证数据的完整性。
  - 继承性，当两个类有大量属性和方法冗余时，一个类（子类）可以从另一个类（父类）继承，从而复用父类的属性和方法，在此基础上再声明自己额外的属性和方法，以减少代码冗余，提升可扩展性。
  - 多态性，指动态绑定（dynamic binding），是指在执行期间判断所引用对象的实际类型，根据其实际的类型调用其相应的方法。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

声明式语言

（1）函数式语言

- Lisp 语言是一种函数式语言，其最显著的特点是程序由表达式组成，而不是由语句组成。
- Lisp 语言的函数可以逐层嵌套，形成复合函数。

```
函数n(...函数2(函数1(数据))...)
```

```
1  (De factorial (n)
2    (Cond ((Eqn n 0) 1)
3          (T (* n ( factorial (- n 1))))))
```

$$f(n) = \begin{cases} 1, & n = 0 \\ n * f(n-1), & n > 0 \end{cases} \tag{1.1}$$

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

声明式语言

（2）基于规则的语言

- Prolog 是一种基于规则的语言，其执行过程是：检查一定的条件，当它满足某个条件，就执行某个动作。

```
条件1 → 动作1
条件2 → 动作2
……
条件n → 动作n
```

```
1  factorial(0,Result) :-
2    Result is 1.
3  factorial(N,Result) :-
4    N > 0,
5    N1 is N-1,
6    factorial(N1,Result1),
7    Result is Result1*N.
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

声明式语言

- （3）数据库查询语言
  - 数据库中的SQL（Structured Query Language）是一种高度非过程化的语言，用户只需要提出“干什么”，无须具体指明“怎么干”，像具体处理操作、存取路径选择等均由系统自动完成。
  - 存储过程则是由SQL语句组成的命令式语言。

```
1  Select * From A where b = 7
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

声明式语言

（4）正则表达式

- 正则表达式（Regular Expression）又称为正规式，用来表达组成文本的规则。

```
1  [A-Za-z_][A-Za-z0-9_]*
2  [\u4E00-\u9FA5A-Za-z_][\u4E00-\u9FA5A-Za-z0-9_]*
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

声明式语言

- （5）组态语言与图形化语言
  - 组态（Configuration）也即配置、设定，指用户通过类似“搭积木”的简单方式来完成自己所需要的软件功能，而不需要编写计算机程序。
  - 图形化语言具有清晰、直观的优点，如软件工程中的统一建模语言UML（Unified Modeling Language）就采用图形表示语义。

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.2.2.1 面向过程的语言

- C 语言是一种典型的命令式语言，其过程是相互独立的，其变量的作用域（Scope）规则为：
  - 在函数体外声明的变量为全局变量，变量声明后的任何函数都可以访问该变量。
  - 函数体内声明的变量为局部变量，仅该函数内的代码可以访问。
  - 函数的形式参数，也只有本函数代码可以访问。

```
1   int a;
2   float f1(float x, float y)
3   {
4     ...
5   }
6   void f2()
7   {
8     int b;
9     ...
10  }
11  int main()
12  {
13    ...
14  }
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.2.2.2 嵌套定义的语言

- Pascal、JavaScript 都是可以嵌套定义过程的语言：
  - 一个过程中声明的变量，对本过程有效。
  - 一个过程中声明的变量，如果在其子过程中未声明，则该变量在该子过程中有效；
  - 如果其子过程对该变量重新声明，则该子过程访问的变量为重新声明过的变量。

```
1  program main;
2    var a: integer;
3    procedure a1(i, j: integer);
4      var b: real;
5      function a11(x, y: real) : real;
6        begin
7          ...
8        end {a11};
9      begin
10       ...
11     end {a1};
12   function a2(x, y: real) : real;
13     begin
14       ...
15     end {a2};
16   begin
17     ...
18   end {main}.
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.2.2.3 面向对象的语言

- C# 是一种典型的面向对象语言：
  - 支持命名空间（包）的概念，类放入到命名空间里面；
  - 使用using引用命名空间。

```
1  using System;
2  ...
3  namespace PersonObjects
4  {
5    public class Person
6    {
7      private string Name;
8      private int Age;
9      ...
10     public Person() {...} // 与类同名的为构造函数
11     public void Go(int nStep)
12     {
13       ...
14     }
15     ...
16   }
17   public class Student : Person
18   {
19     private string No; // 学号
20     ...
21   }
22 }
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.2.2.4 动态语言

- Python 是一种动态语言：
  - 可以在实例化的对象中添加属性和方法，而不必一开始就在类中定义好；
  - 也可以为类添加属性和方法，这样会影响所有创建的对象。

```
1  class Person:
2    def __init__(self, name, age): # 构造函数
3      self.name = name
4      self.age = age
5
6  p1 = Person("骑着大鹅来兜风", 7)
7  # 对象加属性
8  p1.address = "琼楼玉宇"
9  # 类加属性
10 Person.No = 202205;
11 ...
12
13 # 添加成员方法
14 def happy(self):
15   print("蓦然回首，那人却在灯火阑珊处！")
16
17 p2 = Person("最后一只恐龙", 7)
18 p2.study = types.MethodType(study, p)
19 p2.study()
```

山东大学 郑艳伟，zhengyw@sdu.edu.cn

1.2.2.5 基于对象的语言

- JavaScrip 没有类的概念，是一种基于对象的语言（Object-Based Language）：
  - 直接创建模式，直接指定对象的属性和值；
  - 工厂模式，参考函数模式，只是没有this 指针；
  - 构造函数模式，函数相当于类的概念，直接通过构造函数创建对象，使得对象的结构可以复用；
  - Prototype 原型模式，可以通过 Prototype 添加方法。

```
1  // 直接创建对象
2  var PersonX = {
3    name: '骑着大鹅来兜风', // 名字
4    age: 7,        // 年龄
5  }
6  alert(PersonX.name); // 骑着大鹅来兜风
7
8  // 构造函数模式
9  function CreatePerson2(name, age){ // 构造函数首字母大写
10   this.name = name; // 添加属性
11   this.age = age;  // 添加属性
12 }
13 // 实例化
14 var x2 = new CreatePerson2("骑着大鹅来兜风", "7");
15 var y2 = new CreatePerson2("最后一只恐龙", "7");
16 alert(x2.name); // 骑着大鹅来兜风
17
18 // Prototype原型
19 function CreatePerson3(name, age) {
20   this.name = name;
21   this.age = age;
22 }
23 CreatePerson3.prototype.showName = function(){
24   alert(this.name);
25 }
26 //生成实例。
27 var x3 = new CreatePerson2("骑着大鹅来兜风", "7");
28 var y3 = new CreatePerson2("最后一只恐龙", "7");
29 x3.showName(); // 骑着大鹅来兜风
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.2.2.5 基于对象的语言

- Go语言：
  - 通过结构体构造对象；
  - 方法是一种特殊类型的函数，该函数作用在接收者（receiver）上，接收者是某种类型的变量。

```
1  type Person struct {
2    Name string
3    Age  int
4  }
5
6  func (e *Person) ToString() string {
7    return "name=" + e.Name + ";␣age=" + strconv.Itoa(e.Age)
8  }
```

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

基本数据类型

关键字	汇编	说明	字宽	取值范围
byte	sbyte	8 位有符号整型	1	$-128 \sim 127$
ubyte	byte	8 位无符号整型	1	$0 \sim 255$
char	byte	字符型，等价于 ubyte	1	$0 \sim 255$
bool	byte	布尔型，0 为假，非 0 为真	1	$\{true, false\}$
short	sword	16 位有符号整型	2	$-32,768 \sim 32,767$
ushort	word	16 位无符号整型	2	$0 \sim 65,535$
int	sdword	32 位有符号整型	4	$-2^{31} \sim 2^{31}-1$
uint	dword	32 位无符号整型	4	$0 \sim 2^{32}-1$
long	qword	64 位整型	8	–
–	tbyte	80 位整型	10	–
float	real4	32 位 IEEE 短实数，有效数字 6 位	4	$1.18 \times 10^{-38} \sim 3.40 \times 10^{38}$
double	real8	64 位 IEEE 短实数，有效数字 15 位	8	$2.23 \times 10^{-308} \sim 1.79 \times 10^{308}$
real	real10	80 位 IEEE 短实数，有效数字 19 位	10	$3.37 \times 10^{-4932} \sim 1.18 \times 10^{4932}$
*		32 位指针，可以指向任何类型	4	$0 \sim 2^{32}-1$

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

数组

```
1  int a[10, 20];
2  a[2, 3] = 5;
```

结构体

```
1  struct STUDENT
2  {
3    char name[20]; // 姓名
4    uint num;    //学号
5    bool sex;    //性别
6  };
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

（1）表达式

- 表达式递归定义：

① 变量和常数是表达式。

② 如果E_1, E_2是表达式，θ是一个双目运算符，则$E_1 \theta E_2$是表达式。

③ 如果E是表达式，θ是一个单目运算符，则θE（或$E\theta$）是表达式。

④ 如果E是表达式，则(E)是表达式。

⑤ 以上运算经过有限次复合形成的是表达式。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

（1）表达式

- 表达式中的算符：

类别	算符	符号	结合性
位运算符	按位取反	~	右结合
算术算符	幂、负	**(^)、-(@)	右结合
	乘除、取余	*(×), /(÷), %	左结合
	加减	+, -	左结合
位运算符	左移、右移	<<, >>	左结合
关系算符	大于小于	<, ≤ (<=), >, ≥ (>=)	不可结合
	等于不等	==, ≠ (!=, <>)	不可结合
位运算符	按位与	&	左结合
	按位异或	⊕ 或 ^	左结合
	按位或	|	左结合
逻辑算符	非	¬, ! 或 not	右结合
	与	∧, && 或 and	左结合
	或	∨, || 或 or	左结合
三目算符	三目算符	?:	右结合
赋值算符	赋值等	= 或:=	右结合

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

（2）赋值语句

- 赋值语句中的变量（名字）：
  - 运行时的存储单元，称为该名字的左值；
  - 运行时的值，称为该名字的右值。

```
1  z = x;
2  z = x + y;
```

- 并不是所有元素都持有左值和右值，没有左值的元素不能被赋值：
  - 变量既持有左值又持有右值；
  - 常量只持有右值；
  - 表达式只持有右值；
  - 指针变量的右值是一个存储单元，因此指针变量既持有左值又持有右值。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

（3）声明语句

```
1  int a, b; // C语言的声明语句
```

```
1  var a, b: integer; {Pascal语言的声明语句}
```

（4）无条件转移语句

```
1  goto L;
2   ...
3  L:
4   ...
5  goto L;
6   ...
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

（5）if语句

```
1  // C风格的if语句
2  if (B) S
3  if (B) S1 else S2
```

```
1  {Pascal风格的if语句}
2  if B then S
3  if B then S1 else S2
```

（6）三目运算符

```
1  B ? E1 : E2
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

（7）switch语句

```
1  switch (E)
2  {
3    case 0: ...
4    case 1: ...
5    ...
6    case n: ...
7    default: ...
8  }
```

（8）while循环

```
1  while (B) S
2  while B do S
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

（9）for循环

```
1  for (int i = 0; i < n; i++) S
2  for (i = 1 : n) S
3  for (i = 1 : step : n) S
```

（10）foreach循环

```
1  foreach (x in Set) S
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

（11）语句块

- C、C++、C#、Java等语言将语句块用花括号{...}包起来；
- Pascal等语言用begin...end包起来；
- Python采用相同数量的空白确定一个语句块。

（12）函数

```
1  int A(int x, float y)
2  {
3    ...
4    return n;
5  }
6  void B()
7  {
8    int i = 0;
9    float u = 3.14;
10   A(i, u);
11 }
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

1.3 目标语言模型

目标代码的形式

- 绝对指令代码：是能够立即执行的机器语言代码，所有地址均已定位。
- 可重定位指令代码：这是目前常见的形式，其地址是一个相对位置，需要重新定位。
- 汇编指令代码：需经过汇编程序汇编，转换为可执行的机器语言代码。但汇编代码可以使代码生成的过程变的容易，且可读性好，所以我们将汇编代码作为本教材的目标语言。

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

1.3.1 CPU架构和指令集

x86架构

- 1978 年6 月8 日，Intel 发布了16 位微处理器8086，缩写为x86。
- x86 采用CISC 指令集，Intel 官方文档里面称为“IA-32”。
- 采用x86的厂商主要包括：Intel、AMD。
- 发展到64 位机器时，AMD 率先制造出了商用的兼容x86 的CPU，称为AMD64。
- Intel 首先设计了一种不兼容x86 的全新64 位指令集，称之为 “IA-64”。
- 后来Intel 也开始支持AMD64 的指令集，但称为x86-64。
- 因此，当指Intel 的CPU 架构或指令集时，我们仍然称之为x86 系统或x86 指令集；当需要区分64 位和32 位之前的机器时，分别称为x64 和x86。
- 早期的8086 CPU 只能计算整数，浮点数的运算需要一个协处理芯片8087 完成。到80486 时，浮点数的协处理芯片才集成到CPU 芯片上，但两者仍然使用不同的寄存器和计算部件，处理整数和浮点数的部件分别称为CPU 和FPU。

1.3.1 CPU架构和指令集

ARM架构

- ARM 架构是一个精简指令集处理器架构，主要面向嵌入式领域。
- 采用ARM 的厂商主要包括：华为、苹果、谷歌、IBM 等。
- ARM是一款嵌入式处理器，是英国先进RISC机器公司（Advanced RISC Machines，ARM）的产品。
- 该公司不生产具体芯片，只采用IP（Intellectual Property，知识产权）授权的方式允许半导体公司生产基于ARM的处理器产品。
- ARM7基于冯·诺依曼体系结构，ARM9基于哈佛体系结构。
- ARM共有37个32位寄存器，其中31个通用寄存器，用R加一个数字表示。ARM的运算指令一般有3个操作数，如“ADD R1, R0, R3”，表示R0的值加上R3的值，存入到R1。

1.3.1 CPU架构和指令集

RISC-V架构

- RISC-V 架构是基于精简指令集计算原理建立的开放指令集架构（ISA，Open Instruction SetArchitecture），由RISC-V 基金会运营。
- 采用RISC-V的厂商包括：三星、英伟达、西部数据。
- 2013 年，RISC-V 使用BSD（Berkeley Software Distribution）协议开源，这意味着几乎任何人都可以使用RISC-V 指令集进行芯片设计和开发，商品化之后也不需要支付授权费用，因此得到了很多厂商、高校和研究机构的关注。

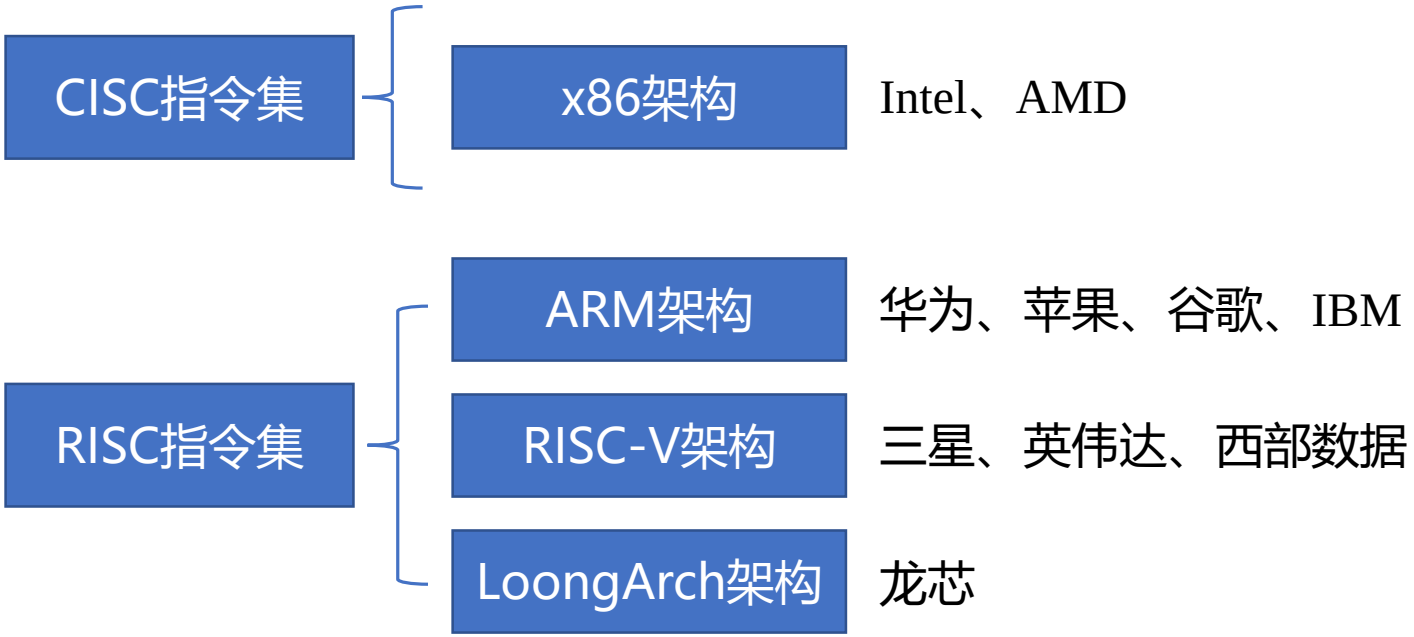
1.3.1 CPU架构和指令集

MIPS和LoongArch架构

- MIPS 架构是一种采用精简指令集的处理器架构，1981 年出现，由MIPS 科技公司开发并授权。
- 它是基于一种固定长度的定期编码指令集，主要采用厂商为龙芯。
- 2020 年，龙芯发布了LoongArch 架构。
- LoongArch 是全新的指令集，并非基于MIPS 的扩展。
- MIPS 只有3 种指令格式，LoongArch 重新设计了指令格式，使可用的格式多达10 种，其包含3种无立即数格式和7 种有立即数格式。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.3.1 CPU架构和指令集



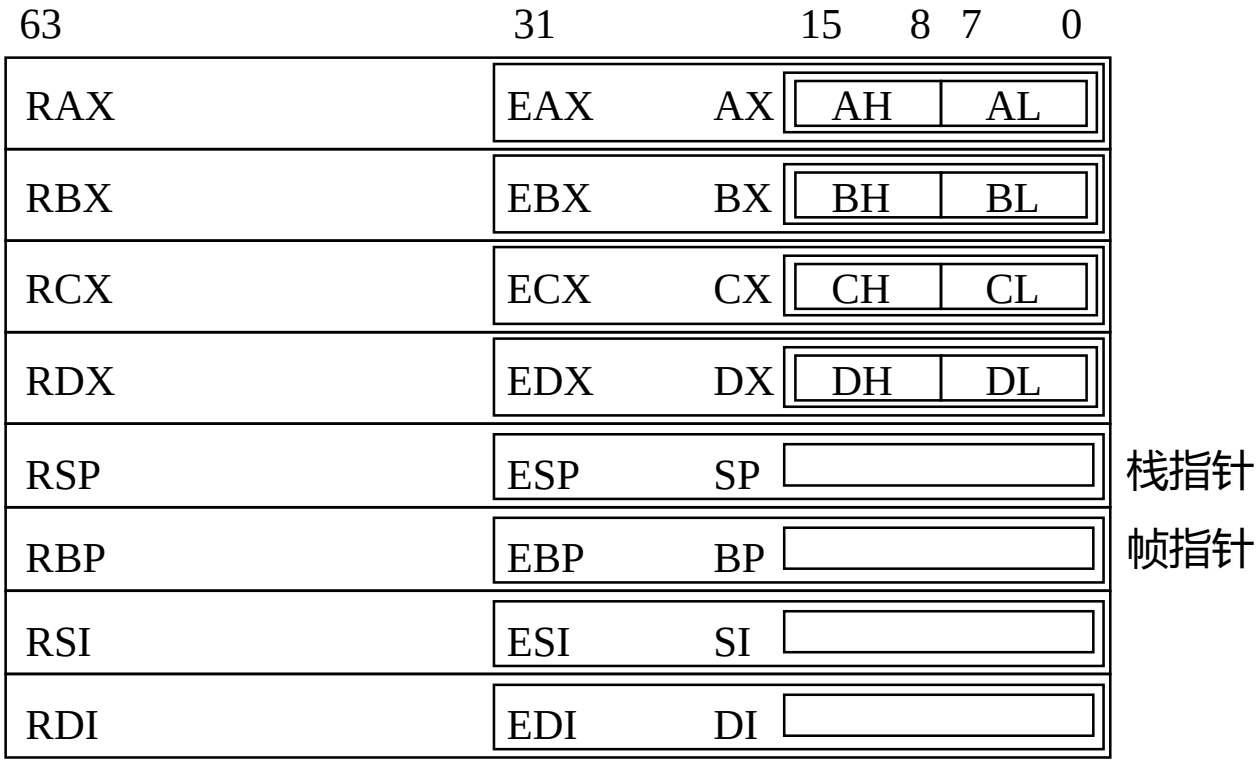
本书以x86为主，兼顾RISC指令集；以32位为主，兼顾64位

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.3.2 寄存器

- 8个通用寄存器
  - 算术运算寄存器EAX、EBX、ECX和EDX，其中ECX也专用做循环控制。
  - 栈操作寄存器ESP自动指向栈顶，EBP一般用于指向帧底。
  - 扩展源变址寄存器ESI和扩展目的变址寄存器EDI一般用于串操作指令。

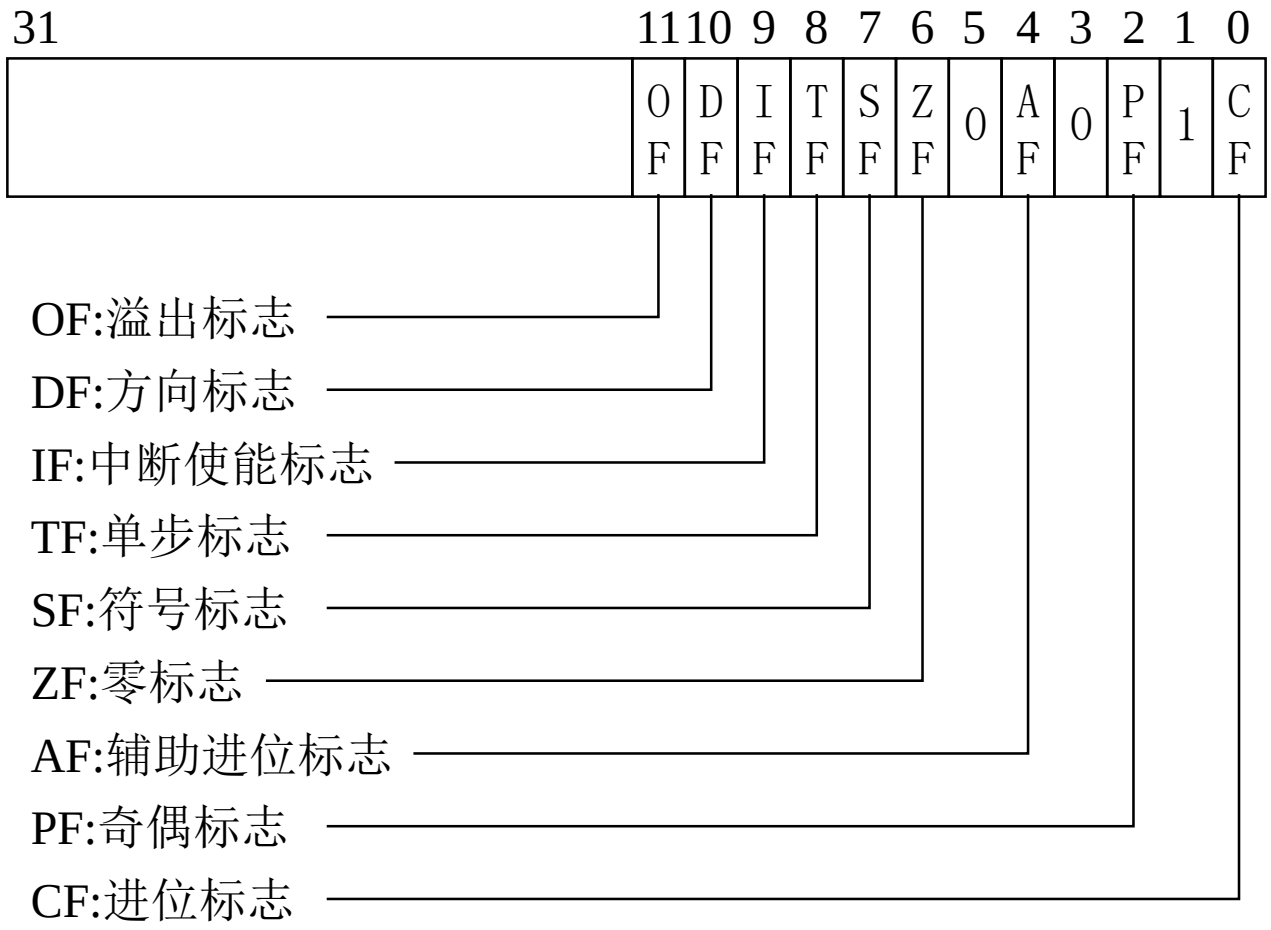


山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.3.2 寄存器

- 6个段寄存器：代码段CS、堆栈段SS、数据段DS、扩展段ES、以及FS和GS
  - 段寄存器是16位
  - 32位环境下基本上不需要修改
  - 64位环境则都是0
- 1个标志寄存器

  EFLAGS



1.3.2 寄存器

- 8个浮点数寄存器
  - 在FPU中，记作ST(0)~ST(n)；
  - 栈顶总是ST(0)，所以弹出一个操作数后，原来的ST(1)变成了ST(0)。
- FPU有6个专用寄存器
  - 浮点数的比较指令影响状态寄存器中的状态字；
  - 转移指令是根据CPU中的EFLAGS寄存器跳转的，因此需要将状态字写入到EFLAGS寄存器的对应位，然后才能正确转移。

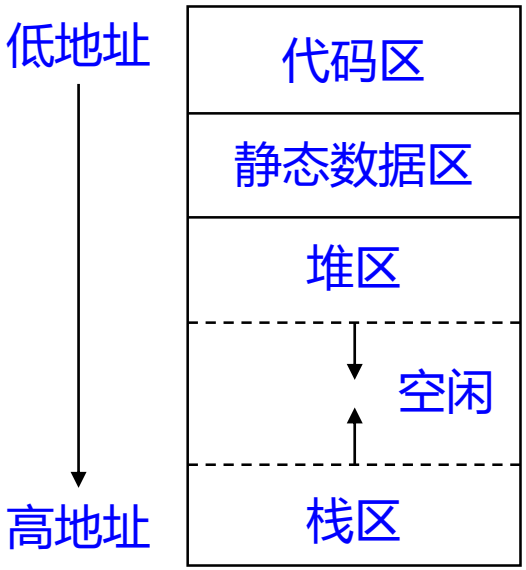
山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.3.3 汇编程序结构

- 运行时内存中的一个程序大致可以分为4个区：
  - 代码区：存放编译器生成的指令代码，以机器指令的形式存在，由CPU 一条条读取并执行。
  - 数据区：存放静态变量。
  - 栈区：主要用来存放函数中的参数、局部变量等，进入一个函数时由编译器申请，函数返回后由编译器自动的释放掉。
  - 堆区：主要用来存储用户动态的申请的临时空间。



1.3.3 汇编程序结构

```
1   .386            ; 伪指令，表示是32位程序
2   .model flat, stdcall  ; 内存模式flat，子程序调用规范stdcall
3   .stack 4096           ; 堆栈大小4096
4
5   ; 标准Windows服务ExitProcess，需要一个退出码做参数
6   ExitProcess PROTO, dwExitCode: DWORD
7
8   .data       ; 数据区
9   sum DWORD 0
10
11  .code       ; 代码区
12  main PROC   ; 过程名（函数名）
13    mov eax, 7
14    add eax, 70
15    mov sum, eax
16
17    invoke ExitProcess, 0  ; 调用操作系统的标准退出服务
18  main ENDP             ; 函数结束
19
20  END main              ; 程序结束，end后面跟入口地址，再往后的内容会忽略
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

1.3.4 汇编指令

汇编语言的标识符（Identifier）规则

- 可以包含1~247个字符，因此编译器最好对标识符的长度进行限定。
- 不区分大小写，因此对大小写敏感的编译器，需要检查是否有两个变量仅仅是大小写不同，如果是则需要修改其中一个。
- 第一个字符必须为大小写字母、下划线、@、?或$，其后的字符除了这些外还可以是数字。
- 标识符不能与汇编器的保留字相同。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

1.3.5 寻址方式及记号约定

记号约定

- imm：指常数，在汇编中称为立即数，如果需要，可以用imm8、imm16、imm32等指定立即数的长度。
- reg：指寄存器，如果需要，可以用reg8、reg16、reg32等指定寄存器的长度。
- mem：指内存地址，如果需要，可以用mem8、mem16、mem32等指定内存数据的长度。

x86有7种寻址方式，编译器使用如下4种：

- addr：addr是一个变量名，直接用该变量名代表的地址寻址。
- [reg]：用寄存器的内容作为地址寻址。
- const[reg]：寄存器的内容加上const作为地址，进行间接寻址。
- [reg + const]：寄存器的内容加上const作为地址，等价于const[reg]。

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.3.6 传送指令

- MOV指令：第一个操作数是目的操作数，第二个操作数是源操作数。
  - MOV reg, imm/reg/mem
  - MOV mem, imm/reg
- LEA指令：Load Effective Address
  - LEA eax, [1000h]
  - LEA ecx, dword ptr [ebx]
  - LEA ebx, x
  - LEA edx, [ebp + 32]
    - ✓ mov edx, ebp
    - ✓ add edx, 32

```
1   .data
2   x    dword 777
3   y    dword ?
4
5   .code
6   mov ebp, esp
7   mov eax, 0A000h
8   mov eax, x
9
10  mov y, eax
11  mov dword ptr x, 5678h
12
13  mov esi,  offset  x
14  mov eax, [esi  + 4]
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

加减指令

加减法相关指令

- ADD reg, imm/reg/mem：两个操作数相加，结果存入第一个操作数的寄存器。
- SUB reg, imm/reg/mem：减法。
- INC reg：自增1，相当于寄存器中的数字加1。
- DEC reg：自减1。
- NEG reg：取反，即一元负运算。

补充说明

- 现在的CPU是有Add mem, reg、Inc mem、Neg mem这样的指令的。
- 此类指令需要将数据从内存取出，使用相应的计算部件计算后再写回，使用的时钟周期较多。
- 程序员当然可以使用这样的指令编程，但对于编译器来说，这些不可分割的操作不利于优化的处理，因此我们不考虑这样的指令。

乘法指令

- 单操作数乘法指令
  - MUL reg/mem；无符号乘法；如果乘积的高半部分不为零，则 MUL 会把进位标志位和溢出标志位置 1。
  - IMUL reg/mem；有符号乘法；如果乘积的高半部分不是其低半部分的符号扩展，则进位标志位和溢出标志位置 1。
- 单操作数乘法指令只有一个操作数，为乘数，另外一个操作数被乘数在一个固定寄存器中，乘积也放在固定的寄存器中
  - 乘法指令操作数为reg8/mem8，被乘数在AL中，乘积存入AX。
  - 乘法指令操作数为reg16/mem16，被乘数在AX中，乘积存入DX:AX；即高16位在DX中，低16位在AX中。
  - 乘法指令操作数为reg32/mem32，被乘数在EAX中，乘积存入EDX:EAX；即高32位在EDX中，低32位在EAX中。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

乘法指令

- 多操作数乘法指令
  - IMUL reg, imm/reg/mem；有符号双操作数乘法，第一个操作数必须是寄存器，乘法结果放在第一个操作数。
  - IMUL reg, imm/reg/mem, imm；有符号三操作数乘法，第一个操作数必须是寄存器，第三个操作数必须是立即数，乘法结果放在第一个操作数。
- 双操作数乘法指令优点
  - 不必将一个操作数放到固定寄存器EAX，可以使用任何寄存器。
  - 计算结果放到第一个操作数，也不需要固定寄存器EAX 和EDX。
  - 第二个操作数可以是立即数、寄存器或内存地址，而单操作数指令必须使用寄存器或内存地址，不能使用立即数乘法。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

乘法指令

```
1   .code
2   ; 初始EAX = 00D7F7E0，EDX = 008F100A，为随机值
3   mov ax, 2002h
4   mov bx, 20h
5   mul bx          ; EFLAGS = 00000A03, EAX = 00D70040, EDX = 008F0004
6
7   mov al, 4
8   mov bl, 4
9   imul bl         ; EFLAGS = 00000202, EAX = 00D70010, EDX = 008F0004
10
11  mov al, -4
12  imul bl         ; EFLAGS = 00000286, EAX = 00D7FFF0, EDX = 008F0004
13
14  mov ax, -4
15  mov bx, 4
16  imul ax, bx     ; EFLAGS = 00000286, EAX = 00D7FFF0, EDX = 008F0004
```

除法指令

除法指令

- DIV reg/mem；无符号除法。
- IDIV reg/mem；有符号除法。

被除数、商和余数使用固定寄存器规则

- 除法指令操作数为reg8/mem8，被除数放入AX，商放入AL，余数放入AH。
- 除法指令操作数为reg16/mem16，被除数放入DX:AX，商放入AX，余数放入DX
- 除法指令操作数为reg32/mem32，被除数放入EDX:EAX，商放入EAX，余数放入EDX。

符号扩展指令

- CBW；将AL的符号位扩展到AH，调用IDiv reg8/mem8前使用。
- CWD；将AX的符号位扩展到DX，调用IDiv reg16/mem16前使用。
- CDQ：将EAX的符号位扩展到EDX，调用IDiv reg32/mem32前使用。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

除法指令

```
1  .code
2  mov eax, -10     ; EAX = FFFFFFF5, EDX = 00330004
3  mov ebx, 4       ; EBX = 00000004
4  cdq              ; EAX = FFFFFFF5, EDX = FFFFFFFF
5  idiv  ebx        ; EAX = FFFFFFFE, EDX = FFFFFFFD, 商-2余-3
```

位操作指令

位操作指令

- SHL reg, imm8/CL；左移，最低位补0，最高位移入EFLAGS寄存器的CF。第一个操作数是目的操作数，第二个是移位次数。
- SHR reg, imm8/CL；右移，最高位补0，最低位移入CF。
- AND reg, imm/reg/mem；按位与。
- OR reg, imm/reg/mem；按位或。
- NOT reg, imm/reg/mem；按位取反。
- XOR reg, imm/reg/mem；按位异或。

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

无条件转移指令

无条件转移指令

- 转移指令都需要一个目的地址，一般用标号实现，标号后面的第一条语句即跳转的目的地址。
- 标号作用与变量名相同，只不过在代码区，且后面加冒号。
- 无条件转移指令为JMP。
- 标号可以在转移指令之前，也可以在之后。

```
1   jmp L1
2    ...
3   L1:
4   mov ...
5    ...
6   jmp L1
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

条件转移指令

条件转移指令

- 先用CMP destination, source指令比较两个数：destination减去source，并不保存结果，只是影响EFLAGS寄存器的标志位，转移指令根据标志位决定是否转移

跳转条件	无符号跳转	有符号跳转	字母说明
相等跳转	JE		E: Equal
不等跳转	JNE		N: Not
大于跳转	JA / JNBE	JG / JNLE	A: Above
大于等于跳转	JAE / JNB	JGE / JNL	G: Greater than
小于跳转	JB / JNAE	JL / JNGE	B: Below
小于等于跳转	JBE / JNA	JLE / JNG	L: Less than

```
1  mov eax, 5
2  cmp eax, 4  ; eax中为5，因此eax > 4
3  jg L1   ; 大于则跳转
4   ...
5  L1:
6   ...
7  cmp eax, 0
8  jnz L1
```

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

栈操作指令

栈操作指令

- PUSH imm32/reg/mem; 将立即数/寄存器/内压入栈顶，注意立即数只能是32 位
- POP reg/mem; 栈顶元素弹出，送入寄存器/内存。
- PUSHAD; 按照EAX、ECX、EDX、EBX、ESP、EBP、ESI、EDI 的顺序，将所有32 位寄存器依次入栈，其中ESP 是执行该指令之前的值，入栈后ESP 的值会改变。
- POPAD; 按相反顺序出栈。
- PUSHFD; EFLAGS 状态寄存器入栈。
- POPFD; EFLAGS 状态寄存器出栈。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

栈操作指令

```
1   sum proc
2     push eax
3     push edx
4
5     mov eax, 7
6     mov edx, 70
7     add eax, edx
8      ...
9
10    pop edx
11    pop eax
12  sum endp
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

初始化指令

- 在进行浮点计算前，先调用FINT指令初始化，其作用是将FPU控制字设置为037FH，具体来说：
  - 屏蔽了所有浮点异常；
  - 舍入模式设置为最近偶数；
  - 计算精度设置为64位。

浮点数存取指令

浮点数存取指令

- FLD m32fp/m64fp/m80fp/ST(i)；将浮点操作数复制到FPU堆栈栈顶，操作数可以是real4、real8、real10内存操作数或FPU寄存器。
- FILD mem；将16位、32位、64位有符号整数源操作数转换为双精度浮点数，并加载到ST(0)。
- FST m32fp/m64fp/m80fp/ST(i)；将ST(0)保存到操作数指向的内存/堆栈寄存器，ST(0)不弹出堆栈。
- FSTP m32fp/m64fp/m80fp/ST(i)；将ST(0)保存到操作数指向的内存/堆栈寄存器，ST(0)弹出堆栈。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

浮点数存取指令

浮点数存取指令

- FIST mem16/mem32：将ST(0)转换为有符号整数，并把结果保存到目的操作数，保存的值可以是字或双字。默认向上舍入，由控制字的第10~11位控制。如果要改变舍入模式，可以用下面的FSTCW指令把控制字取到内存，然后修改相应控制位后再用FLDCW加载回去。
- FLDCW mem16：从内存加载控制字。
- FSTCW mem16：保存控制字到内存。

```
1  .data
2    array real8 10 dup(?)    ; 10个real8类型的数据
3    One real8 77.7           ; 1个real8数据
4  .code
5    fld One                  ; 直接寻址
6    fld [array + 8]          ; 直接偏移
7    fld real8 ptr [esi]      ; 间接寻址
8    fld array[esi]           ; 变址寻址
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

浮点数存取指令

一些特殊常数加载

- FLDZ：将0.0压入栈顶。
- FLD1：将1.0压入栈顶。
- FLDL2T：将$\log_2 10$压入栈顶。
- FLDL2E：将$\log_2 e = \frac{1}{\ln 2}$压入栈顶。
- FLDPI：将π压入栈顶。
- FLDLG2：将lg2压入栈顶。
- FLDLN2：将ln2压入栈顶。

浮点数计算指令

- 浮点数计算指令的操作高度一致
  - FADD、FSUB、FMUL、FDIV，统一使用op表示。
  - FADDP、FSUBP、FMULP、FDIVP统一用opp表示。
  - FIADD、FISUB、FIMUL、FIDIV统一用fiop表示。
- 浮点数计算指令
  - op; 没有操作数，ST(1) op ST(0)的结果放入ST(1)，然后ST(0)出栈。
  - op m32fp/m64fp; ST(0) op m32fp/m64fp，存入ST(0)。
  - op ST(0), ST(i); ST(0) op ST(i)，存入ST(0)。
  - op ST(i), ST(0); ST(i) op ST(0)，存入ST(i)。
  - opp ST(i), ST(0); ST(i) op ST(0)，存入ST(i)，且ST(0)出栈。
  - fiop m16int/m32int; 将源操作数转换为扩展双精度浮点数，再从ST(0)中fiop该数，存入ST(0)。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

浮点数计算指令

取反和绝对值

- FCHS; ST(0)取反。
- FABS; ST(0)取绝对值。

比较与转移

比较与跳转

- FCOM; 比较ST(0)与ST(1)。
- FCOM m32fp/m64fp; 比较ST(0)与m32fp/m64fp。
- FCOM ST(i); 比较ST(0)与ST(i)。
- FCOMP指令操作数类型与执行的操作与FCOM指令相同，但要将ST(0)弹出堆栈
- FCOMPP指令操作数类型与执行的操作与FCOM指令相同，但有两次出栈操作
- FNSTSW AX指令把FPU状态字送入AX，操作数只能是AX，不能是其他寄存器
- SAHF指令把AH复制到EFLAGS寄存器。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

比较与转移

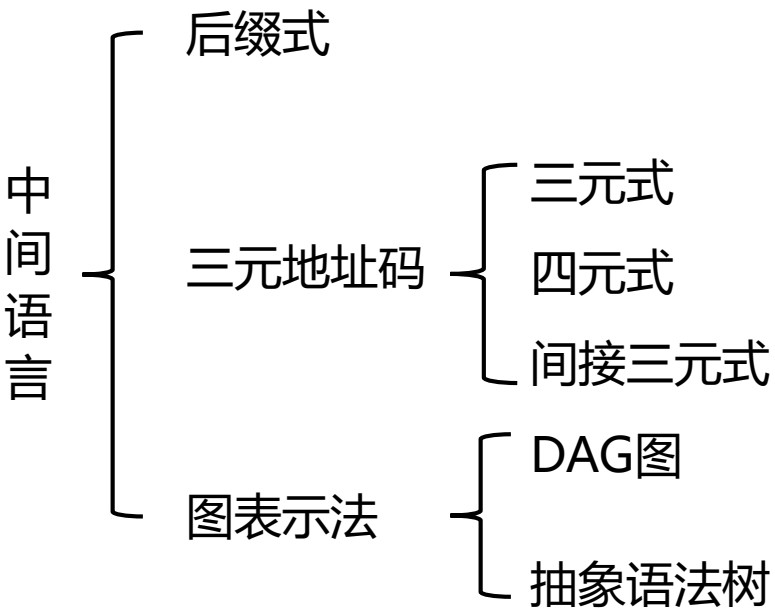
```
1   .data
2     x real8 1.2
3     y real8 3.0
4     n dword 0
5   .code
6     fld x        ; ST(0) = x
7     fcomp y      ; ST(0)和y比较
8     fnstsw ax    ; FPU的状态字送入AX
9     sahf         ; AH复制到EFLAGS
10    jge L1       ; x>=y跳过下条指令
11    mov dword ptr n, 1   ; n = 1
12    L1: ...
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

1.4 中间语言

- 中间语言与具体机器特性无关，是源语言到目标语言的一个过渡
  - 源语言生成中间语言后，生成不同目标机器语言时，只需修改中间代码到目标代码的映射部分，算法结构更清晰。
  - 可对中间语言进行与机器无关的优化，有利于提高目标代码的质量。
  - 考虑m 种高级语言n 种目标机器语言的编译，可极大降低编译器开发工作量。



山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

1.4.1 后缀式

- 后缀式表示法是波兰逻辑学家卢卡西维奇（Lukasiewicz）发明的一种表达式的表示方法，因此又称逆波兰表示法。
- 这种表示法是：把运算量写在前面，把算符写在后面。
  - $a+b$写成$ab+$，$a*b$写成$ab*$
- 用E或E_i表示中缀形式，$<E>$表示E的后缀形式，一个表达式E的后缀形式定义如下
  - ① 如果a是一个变量或常量，$<a>=a$；
  - ② 如果θ是单目运算符，$<\theta E>=<E>\theta$；
  - ③ 如果θ是双目运算符，$<E_1\theta E_2>=<E_1><E_2>\theta$；
  - ④ $<(E)>=<E>$。

1.4.1 后缀式

- 与中缀及前缀表示相比：
  - 运算符个数不变；
  - 运算量的次序和个数不变。
- 后缀式优点：
  - 无括号，形式简洁；
  - 运算符的顺序与运算次序完全相同。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.4.1 后缀式

【例1.1】

$< a * b + c * d >$

$= < a * b >< c * d > +$

$= ab * cd * +$

【例1.2】

$< (a + b) * (c * d + e) >$

$= < (a + b) >< (c * d + e) > *$

$= < a + b >< c * d + e > *$

$= ab + < c * d >< e > + *$

$= ab + cd * e + *$

【例1.3】

$< (a + b)\hat{}(c * d)\hat{}e >$

$= < (a + b) >< (c * d)\hat{}e > \hat{}$

$= < a + b >< (c * d) >< e > \hat{}\hat{}$

$= ab + cd * e\hat{}\hat{}$

【例1.4】

$< a \leq b + c \wedge a > d \vee a + b \neq e >$

$= < a \leq b + c \wedge a > d >< a + b \neq e > \vee$

$= < a \leq b + c >< a > d > \wedge < a + b >< e > \neq \vee$

$= < a >< b + c > \leq ad > \wedge ab + e \neq \vee$

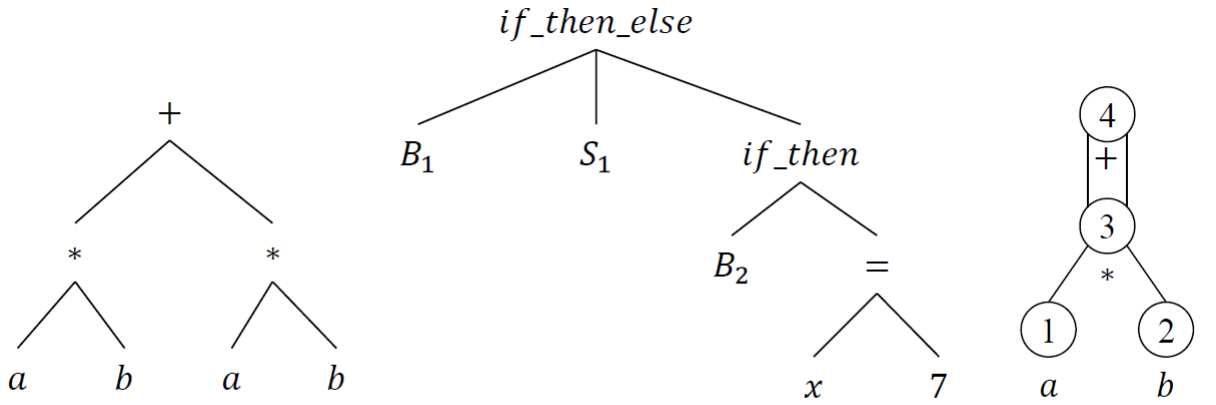
$= abc + \leq ad > \wedge ab + e \neq \vee$

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

1.4.2 图表示法

- 抽象语法树（Abstract Syntax Tree）：是将操作符作为内部结点，操作数作为叶结点构成的树型数据结构。
- DAG图，即无循环有向图（Directed Acyclic Graph）。
  - 可以识别公共子表达式，在代码优化部分介绍。



(a) $a*b+a*b$ 的 AST　(b) if B_1 then S_1 else if B_2 then $x=7$ 的 AST　(c) $a*b+a*b$ 的 DAG

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

1.4.3 三地址代码

- 三地址代码：由以下一般形式的语句构成的序列$x = y\ op\ z$。
  - x, y, z为名字、常数或编译产生的临时变量；
  - op代表运算符号。
- $x + y * z$可以翻译成如下语句序列：

$\$1 = y * z$

$\$2 = x + \$1$

其中\$1和\$2为编译时产生的临时变量。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.4.3 三地址代码

- 四元式：带有四个域的记录结构$(op, arg1, arg2, result)$，相当于$result = arg1\ op\ arg2$
  - op为一个代表运算符的内部码；
  - $arg1, arg2$分别为左操作数和右操作数；
  - $result$结果域。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

类别	三地址码	四元式	说明
一般运算	$z = x\theta y$	(θ, x, y, z)	θ 为二元算符，如算术、关系、逻辑算符等
	$z = \theta x$	$(\theta, x, -, z)$	θ 为一元算符，如一元负、逻辑非等。
	$z = x$	$(=, x, -, z)$	赋值运算。
转移语句	goto L	$(j, -, -, L)$	无条件转移到 L。
	if a goto L	$(jnz, a, -, L)$	a 为真转移到 L，jnz 表示 not zero 时转移。
	if $x\theta y$ goto L	$(j\theta, x, y, L)$	$x\theta y$ 为真转移到 L，θ 是关系运算符，因此 $j\theta$ 是 $j=$、$j\neq$、$j<$、$j\leq$、$j>$、$j\geq$ 之一。
数组	$z = x[i]$	$(=[], x, i, z)$	把地址 x 后面的第 i 个单元的内容赋值给 x。
	$z[i] = x$	$([]=, i, x, z)$	把 x 赋值给地址 z 后面的第 i 个单元。
地址指针	$z = \&x$	$(\&, x, -, z)$	把 x 的地址赋值给 z。
	$z = *x$	$(=*, x, -, z)$	把地址 x 中的内容赋值给 z。
	$*z = x$	$(*=, x, -, z)$	把 x 写入到地址 z。
过程定义	name proc	$(proc, name, -, -)$	过程开始标记，过程名字为 $name$。
	name endp	$(endp, name, -, -)$	过程结束标记，过程名字为 $name$。
过程调用与返回	param x	$(param, x, -, -)$	x 是函数实参，有 n 实参就需要 n 个该语句。
	call p	$(call, p, -, -)$	调用过程 p。
	ret x	$(ret, x, -, -)$	返回 x。
形参标记	$def\ x$	$(def, -, -, x)$	形参 x 定值标记。
寄存器分配专用	$load\ x$	$(load, -, -, x)$	将变量 x 从主存加载到寄存器。
	$store\ x$	$(store, x, -, -)$	将变量 x 从寄存器保存到主存。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.4.3 三地址代码

【例】$a = b * -c + b * -c$的四元式

$(@, c, -, \$1)$

$(*, b, \$1, \$2)$

$(@, c, -, \$3)$

$(*, b, \$3, \$4)$

$(+, \$2, \$4, \$5)$

$(=, \$5, -, a)$

1.4.3 三地址代码

- 三元式：为避免把临时变量填入到符号表，可以通过计算这个临时变量的语句位置来引用这个临时变量，这样只需要三个域$(op, arg1, arg2)$
  - op为一个代表运算符的内部码；
  - $arg1, arg2$为运算数或三元式标号。

【例】$a = b * -c + b * -c$的三元式

$(0)\ (@, c, -)$

$(1)\ (*, b, (0))$

$(2)\ (@, c, -)$

$(3)\ (*, b, (2))$

$(4)\ (+, (1), (3))$

$(5)\ (=, a, (4))$

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.4.3 三地址代码

- 间接三元式：为了便于代码优化处理，有时不直接使用三元式表，而是另设一张指示器（称为间接码表），它将按运算的先后顺序列出有关三元式在三元式表中的位置。
  - 代码优化过程中需要调整运算顺序时，只需重新安排间接码表。

【例】如下语句的间接三元式

$$x = (a + b) * c$$

$$y = d\hat{}(a + b)$$

间接代码	三元式表
(1)	(1) $(+, a, b)$
(2)	(2) $(*, (1), c)$
(3)	(3) $(=, x, (2))$
(1)	(4) $(\hat{}, d, (1))$
(4)	(5) $(=, y, (4))$
(5)	

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

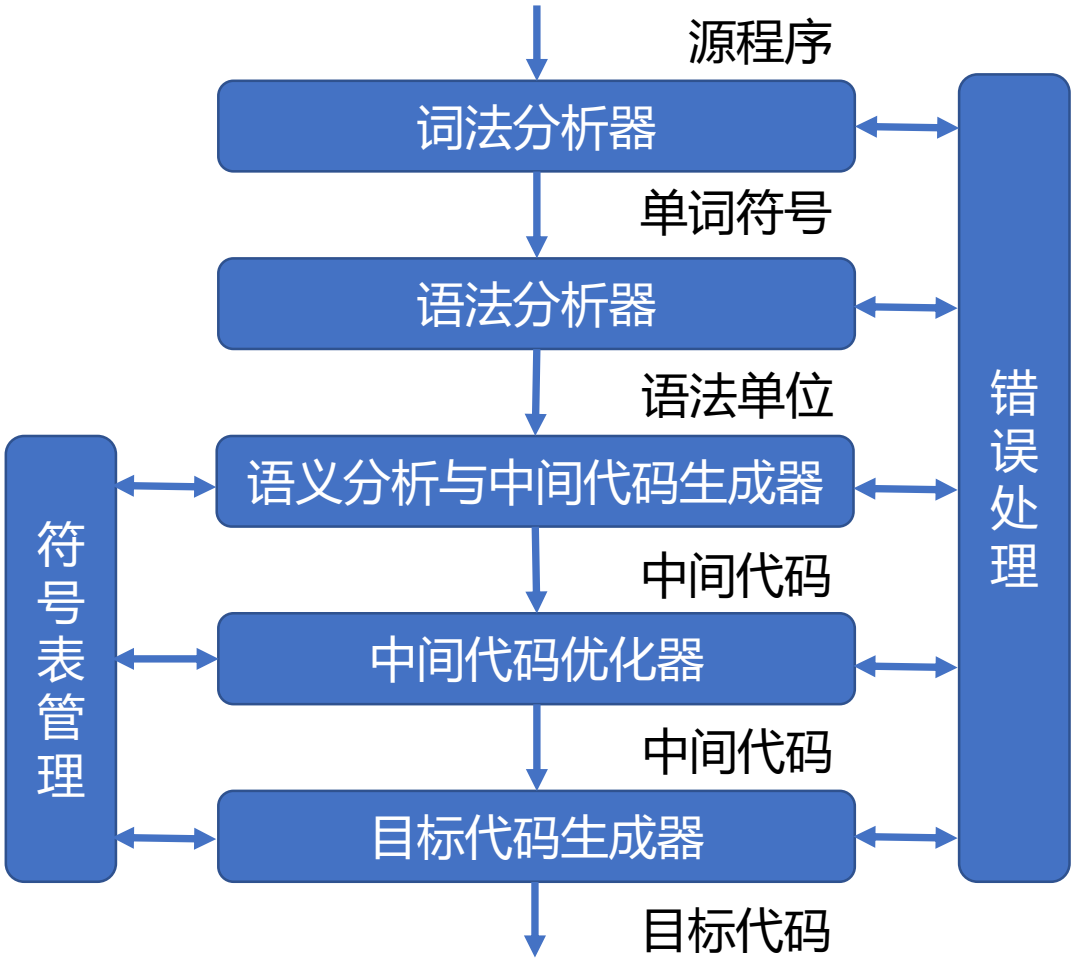
1.5.1 编译器框架

- 自然语言的翻译
  - ① 识别出句子中的一个个单词
  - ② 分析句子的语法结构
  - ③ 根据句子的含义进行初步翻译
  - ④ 对译文进行修饰
  - ⑤ 写出最后的译文

- 编译
  - ① 词法分析
  - ② 语法分析
  - ③ 语义分析和代码自动生成
  - ④ 代码优化
  - ⑤ 目标代码生成

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

词法分析器



- 词法分析器，又称扫描器，输入源程序，进行词法分析，输出单词符号。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

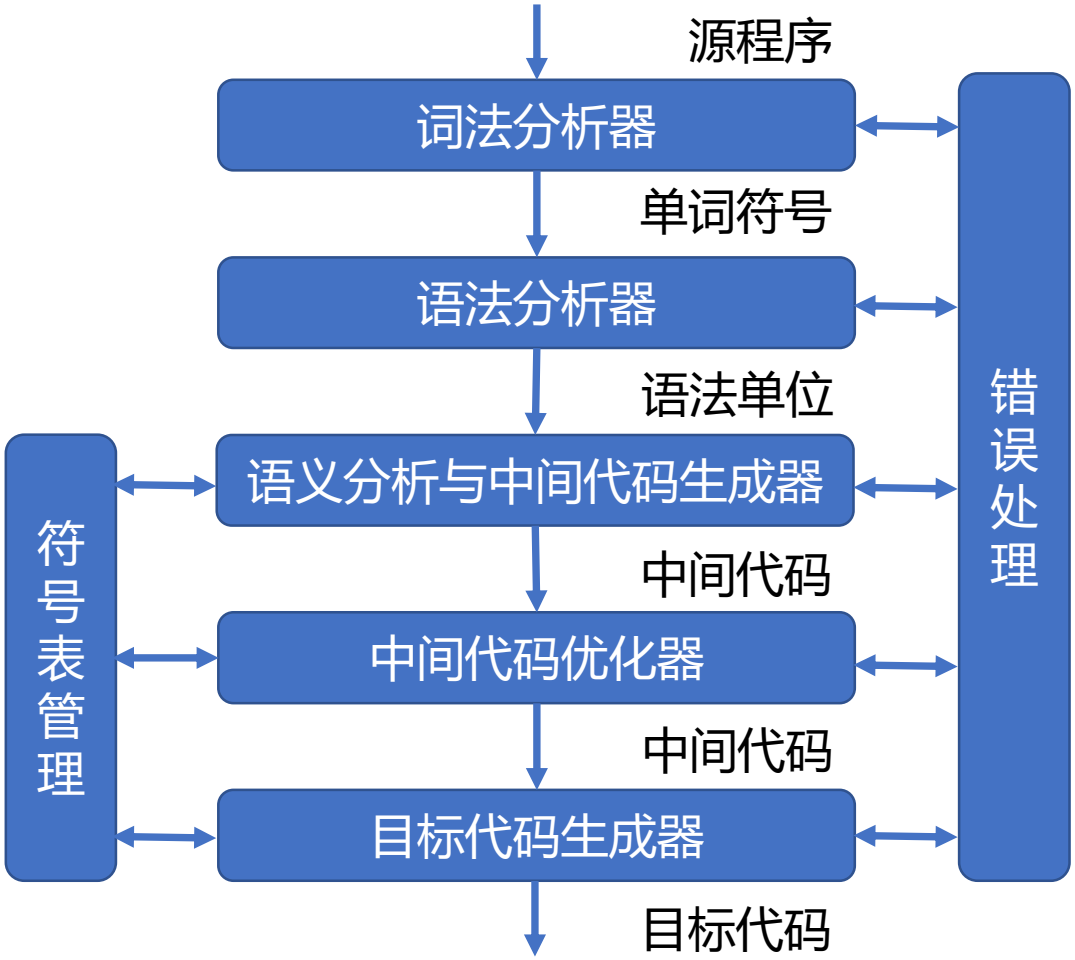
词法分析器

```
1  int Fun(int x, int y) {
2    int a, b, var;
3    a = x + y;
4    b = x + y;
5    var = a * b;
6    return var;
7  }
```

(1) (int, 关键字)
(2) (Fun, 标识符)
(3) ((, 界符)
(4) (int, 关键字)
(5) (x, 标识符)
(6) („ 界符)
(7) (int, 关键字)
(8) (y, 标识符)
(9) (), 界符)
(10) ({, 界符)
(11) (int, 关键字)
(12) (a, 标识符)
(13) („ 界符)
(14) (b, 标识符)
(15) („ 界符)
(16) (var, 标识符)
(17) (;, 界符)
(18) (a, 标识符)
(19) (=, 算符)
(20) (x, 标识符)
(21) (+, 算符)
(22) (y, 标识符)
(23) (;, 界符)
(24) (b, 标识符)
(25) (=, 算符)
(26) (x, 标识符)
(27) (+, 算符)
(28) (y, 标识符)
(29) (;, 界符)
(30) (var, 标识符)
(31) (=, 算符)
(32) (a, 标识符)
(33) (*, 算符)
(34) (b, 标识符)
(35) (;, 界符)
(36) (return, 关键字)
(37) (var, 标识符)
(38) (;, 界符)
(39) (}, 界符)

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

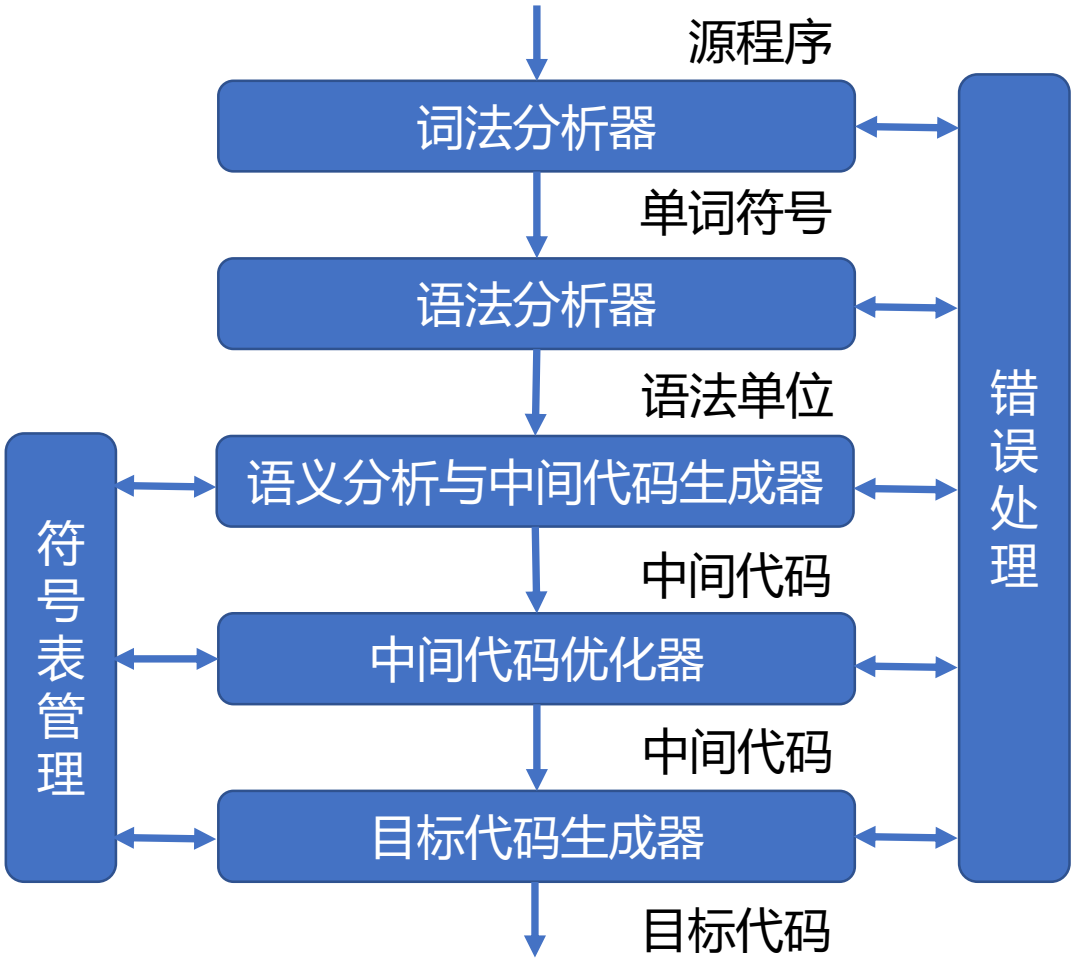
语法分析器



- 语法分析器，又称扫描器，对单词符号串进行语法分析，识别出各类语法单位，最终判断输入串是否构成语法上正确的“程序”。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

语义分析与中间代码生成器



- 语义分析与中间代码生成器，按照语义规则对语法分析器归约（或推导）出的语法单位进行语义分析，并把它们翻译成一定形式的中间代码。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

语义分析与中间代码生成器

```
1  int Fun(int x, int y) {
2    int a, b, var;
3    a = x + y;
4    b = x + y;
5    var = a * b;
6    return var;
7  }
```

(1) (proc, Fun, −, −)
(2) (+, x, y, $1)
(3) (=, $1, −, a)
(4) (+, x, y, $2)
(5) (=, $2, −, b)
(6) (*, a, b, $3)
(7) (=, $3, −, var)
(8) (ret, var, −, −)
(9) (endp, Fun, −, −)

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

符号表

- 符号表：用于登记源程序的各类信息和编译各阶段的进展状况。
  - 符号表：登记源程序中的每个名字及属性，如变量名、常量名、过程名等。
  - 变量名的信息包括：类型、占用内存大小、地址等等。
  - 通常，编译程序在处理到名字的定义性出现时，把名字的各种属性填入符号表；处理到名字的使用性出现时，对名字的属性进行查证。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

符号表

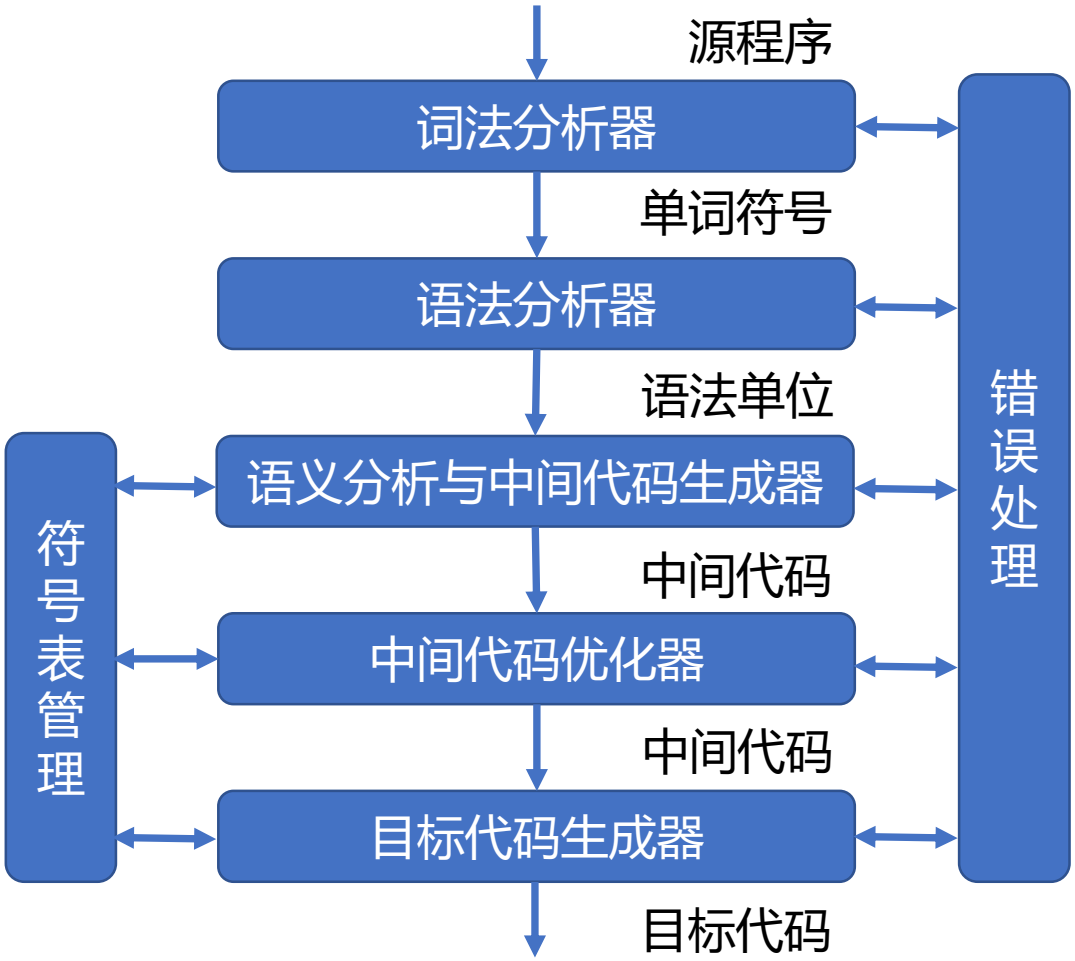
```
1  int Fun(int x, int y) {
2    int a, b, var;
3    a = x + y;
4    b = x + y;
5    var = a * b;
6    return var;
7  }
```

(1) (proc, Fun, −, −)
(2) (+, x, y, $1)
(3) (=, $1, −, a)
(4) (+, x, y, $2)
(5) (=, $2, −, b)
(6) (*, a, b, $3)
(7) (=, $3, −, var)
(8) (ret, var, −, −)
(9) (endp, Fun, −, −)

名字	类别	类型	大小	偏移量
x	形参变量	sdword	4	0
y	形参变量	sdword	4	4
a	局部变量	sdword	4	0
b	局部变量	sdword	4	4
var	局部变量	sdword	4	8
$1	临时变量	sdword	4	−1
$2	临时变量	sdword	4	−1
$3	临时变量	sdword	4	−1

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

中间代码优化器



- 中间代码优化器，对中间代码进行优化处理。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

中间代码优化器

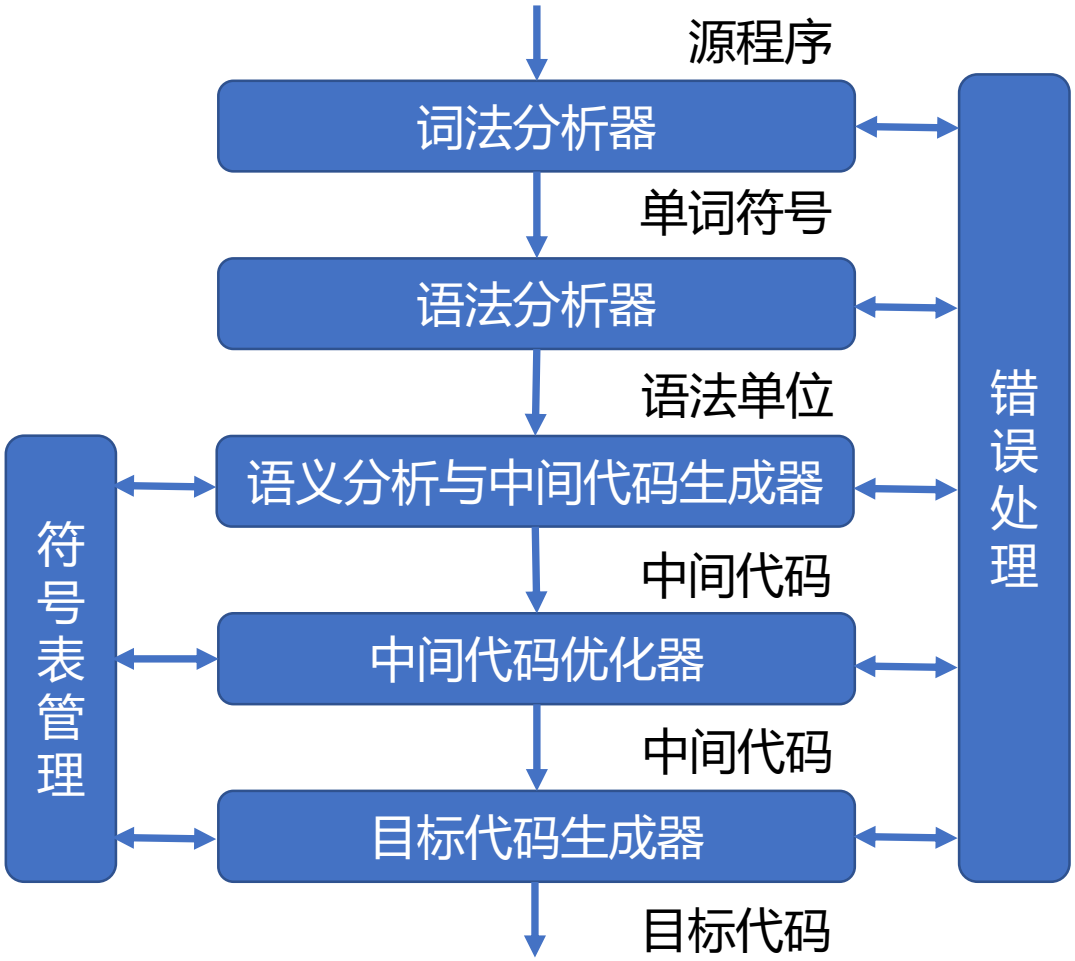
```
1  int Fun(int x, int y) {
2    int a, b, var;
3    a = x + y;
4    b = x + y;
5    var = a * b;
6    return var;
7  }
```

(1) (proc, Fun, −, −)
(2) (+, x, y, $1)
(3) (=, $1, −, a)
(4) (+, x, y, $2)
(5) (=, $2, −, b)
(6) (*, a, b, $3)
(7) (=, $3, −, var)
(8) (ret, var, −, −)
(9) (endp, Fun, −, −)

(1) (proc, Fun, −, −)
(2) (+, x, y, $1)
(3) (*, $1, $1, $3)
(4) (=, $3, −, var)
(5) (ret, var, −, −)
(6) (endp, Fun, −, −)

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

目标代码生成器



- 目标代码生成器，把中间代码翻译成目标代码。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

目标代码生成器

(1) (proc, Fun, −, −)
(2) (+, x, y, $1)
(3) (*, $1, $1, $3)
(4) (=, $3, −, var)
(5) (ret, var, −, −)
(6) (endp, Fun, −, −)

```
(1) Fun proc
(2) push ebp
(3) mov ebp, esp
(4) push ebx
(5) push esi
(6) push edi
(7) sub esp, 12
(8) mov eax, [ebp + 8]
(9) add eax, [ebp + 12]
(10) mov ebx, eax
(11) imul ebx, eax
(12) mov eax, ebx
(13) jmp ?6
(14) ?6:
(15) add esp, 12
(16) pop edi
(17) pop esi
(18) pop ebx
(19) pop ebp
(20) ret 8
(21) Fun endp
```

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

错误处理

错误处理

- 最大限度的发现各种错误。
- 准确指出错误的性质和发生错误的地点。
- 将错误所造成的影响限制在尽可能小的范围，以便使得源程序的其余部分可以继续被编译下去，以进一步发现其它可能的错误。
- 如果可能，自动校正错误。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.5.2 编译前端与后端

- 遍：对源程序或中间结果从头到尾扫描一次，并做有关加工处理，生成新的中间结果或目标程序。
  - 开始于从外存上获得前一遍的中间结果；
  - 结束于完成相关工作并记录于外存。
- 可以几个不同阶段合为一遍，也可以一个阶段分成若干遍。
  - 如词法分析、语法分析合为一遍，甚至与语义分析和中间代码生成一起合为一遍。
  - 又如优化阶段，按优化目标分为多遍。
  - 遍数多则逻辑清晰，但效率差。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.5.2 编译前端与后端

- 前端：由与源语言有关但与目标机器无关的部分组成。
  - 词法分析
  - 语法分析
  - 语义分析与中间代码生成
  - 与目标机无关的代码优化
- 后端：与目标机器有关的部分。
  - 目标代码生成
  - 与目标机有关的代码优化

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

第一章作业

【作业1-1】画图表示编译过程的各个阶段，并简要说明各阶段的功能。

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

1.6.1 数理逻辑和记号

- 与、或、非
  - 公式或算法中，分别采用 ∧、∨、¬ 表示，如 $\neg a \wedge x \leq y$；
  - 代码中，分别采用 &&、||、！表示，如 $!a \;\&\&\; x \leq y$ 。
- 蕴含式和双向蕴含式
  - “如果P则Q”，用 $P \rightarrow Q$表示；
  - “P当且仅当Q”，用 $P \leftrightarrows Q$表示；
  - $P \rightarrow Q$为重言式（永真式），用 $P \Rightarrow Q$表示；
  - $P \leftrightarrows Q$为重言式（永真式），用 $P \Leftrightarrow Q$表示；

1.6.1 数理逻辑和记号

量词

- 全称量词 ∀ 表示“所有的”，如 $(\forall x)(P(x) \rightarrow I(x))$，其中$P(x)$和$I(x)$分别表示$x$是正整数和整数；
- 存在量词 ∃ 表示“存在”，如 $(\exists x)(I(x) \rightarrow P(x))$。

书写约定

- 公式和算法中的变量用斜体，代码中的变量用正体；
- 公式和算法中的符号，一般采用单个字母、希腊字母或缩写形式；
- 代码中的变量，请采用编程命名规范，如匈牙利命名法、驼峰命名法、帕斯卡命名法、下划线命名法等。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

1.6.2 集合论

- 集合：一般把具有共同性质的东西汇聚为一个整体，用花括号括起来表示
  - 列举法：$\Sigma = \{a, b, \ldots, z, 0,1, \ldots, 9\}$；
  - 规则定义法：$\Sigma = \{x \mid 48 \leq ascii(x) \leq 57 \vee 97 \leq ascii(x) \leq 122\}$。
- 集合关系定义
  - $\forall x(x \in A \leftrightarrows x \in B) \rightarrow A = B$
  - $A \subseteq B \Leftrightarrow \forall x(x \in A \rightarrow x \in B)$
  - $A = B \Leftrightarrow A \subseteq B \wedge B \subseteq A$
  - $A \subset B \Leftrightarrow (\forall x)(x \in A \rightarrow x \in B) \wedge (\exists x)(x \in B \wedge x \notin A)$

1.6.2 集合论

集合的运算

- $A \cap B = \{x \mid x \in A \wedge x \in B\}$
- $A \cup B = \{x \mid x \in A \vee x \in B\}$
- $A - B = \{x \mid x \in A \wedge x \notin B\}$
- $\bar{A} = \{x \mid x \in \Omega \wedge x \notin A\}$

集合的几个等价运算

- $A - B = A - A \cap B$
- $A - B = A \cap \bar{B}$
- $\bar{A} = \Omega - A$

例

- $\Omega = \{a, b, \ldots, z, 0,1, \ldots, 9\}$
- $A = \{1,3,5,7,9\}$
- $B = \{2,4,6,7,8\}$
- $C = \{a, b, \ldots, z\}$

有

- $A \cap B = \{7\}$
- $A \cup B = \{1,2,3,4,5,6,7,8,9\}$
- $A - B = \{1,3,5,9\}$
- $\bar{A} = \Omega - A = \{a, b, \ldots, z, 0,2,4,6,8\}$
- $A \cap C = \emptyset$

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

1.6.2 集合论

几个简写运算符

- $A \cap= B \Leftrightarrow A = A \cap B$
- $A \cup= B \Leftrightarrow A = A \cup B$
- $A -= B \Leftrightarrow A = A - B$
- $\bigcap_{i=1}^{n} A_i = A_1 \cap A_2 \cap \cdots \cap A_n$
- $\bigcup_{i=1}^{n} A_i = A_1 \cup A_2 \cup \cdots \cup A_n$

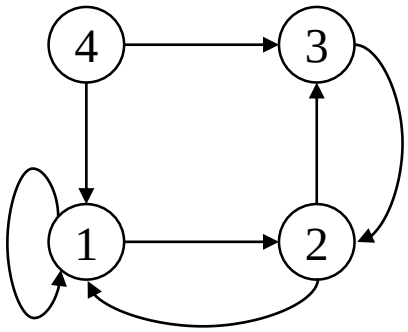
- 1.1 编译器的基本概念
  - 1.1.1 语言的分类
  - 1.1.2 程序设计语言分类
  - 1.1.3 编译程序
  - 1.1.4 编译原理与技术的特点
  - 1.1.5 编译程序的生成
- 1.2 高级程序设计语言
  - 1.2.1 高级语言分类
  - 1.2.2 程序结构
  - 1.2.3 数据类型
  - 1.2.4 语句
- 1.3 目标语言模型
  - 1.3.1 CPU架构和指令集
  - 1.3.2 寄存器
  - 1.3.3 汇编程序结构
  - 1.3.4 汇编指令
  - 1.3.5 寻址方式及记号约定
  - 1.3.6 传送指令
  - 1.3.7 基本运算指令
  - 1.3.8 转移指令
  - 1.3.9 栈操作指令
  - 1.3.10 浮点指令
- 1.4 中间语言
  - 1.4.1 后缀式
  - 1.4.2 图表示法
  - 1.4.3 三地址码
- 1.5 编译器组成
  - 1.5.1 编译器框架
  - 1.5.2 编译前端与后端
- 1.6 数学基础
  - 1.6.1 数理逻辑和记号
  - 1.6.2 集合论
  - 1.6.3 图论

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

邻接矩阵

- 一个图有n个结点，记作$v_1, v_2, \ldots, v_n$，定义其邻接矩阵为$A = (a_{i,j})_{n\times n}$，其中

$$a_{i,j} = \begin{cases} 1, & v_i\text{到}v_j\text{有边} \\ 0, & v_i\text{到}v_j\text{无边} \end{cases}$$



$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

通路数

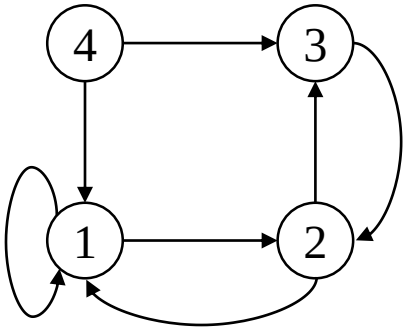
- 考虑结点v_i到结点v_j长度为2的通路数
  - 如果v_i到v_j只有一条长度为2的通路，则必然经过且经过一个结点，假设为v_k，即$v_i \to v_k \to v_j$，那么必然有$a_{i,k}=1, a_{k,j}=1$。
  - 如果v_i到v_j没有长度为2的通路，即$(\nexists k)(v_i \to v_k \to v_j)$，那么要么$a_{i,k}=0$，要么$a_{k,j}=0$。
  - 测试所有n个结点，v_i到v_j长度为2的通路数为$\sum_{k=1}^{n} a_{i,k}a_{k,j}$。
  - 记$A^2=AA=(a_{i,j}^{(2)})_{n\times n}$，则$a_{i,j}^{(2)}$就是结点$v_i$到$v_j$长度为2的通路数。
  - 同理，A^m的每个元素$a_{i,j}^{(m)}$，就是结点v_i到v_j长度为m的通路数。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

邻接矩阵

通路数

- $v_1 \to v_1$，长度为1的路有1条，长度为2的路有2条，长度为3的路有3条；
- $v_2 \to v_4$，长度为1的路有0条，长度为2的路有0条，长度为3的路有0条；
- $v_3 \to v_3$，长度为1的路有0条，长度为2的路有1条，长度为3的路有0条；
- $v_4 \to v_3$，长度为1的路有1条，长度为2的路有0条，长度为3的路有2条。



$$A=\begin{pmatrix}1&1&0&0\\1&0&1&0\\0&1&0&0\\1&0&1&0\end{pmatrix},A^2=\begin{pmatrix}2&1&0&0\\1&2&0&0\\1&0&1&0\\1&2&0&0\end{pmatrix},A^3=\begin{pmatrix}3&3&1&0\\3&1&2&0\\1&2&0&0\\3&1&2&0\end{pmatrix}$$

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

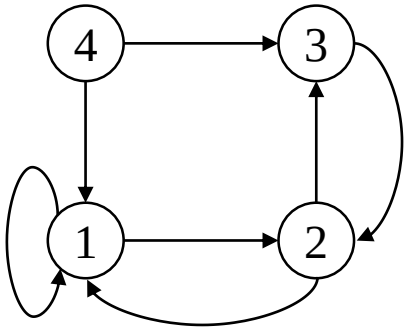
可达矩阵

- 一个图有n个结点，记作$v_1, v_2, \ldots, v_n$，定义其可达矩阵为$R = (r_{i,j})_{n\times n}$，其中

$$r_{i,j} = \begin{cases} 1, & a_{i,j} > 0 \\ 0, & a_{i,j} = 0 \end{cases}$$

- 可达矩阵

$$R = R^{(1)} \vee R^{(2)} \vee \cdots = \bigvee_{k=1}^{+\infty} R^{(k)}$$



$$R = \bigvee_{k=1}^{4} R^{(k)} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

第 1 章 编译器概述 内容小结

- 编译器研究如何将高级程序设计语言翻译为低级目标语言。
- 高级程序设计语言分为命令式语言和声明式语言，按程序结构可分为面向过程语言、过程嵌套语言、面向对象语言、动态语言和基于对象语言。
- CPU 架构包括 x86、ARM、RISC-V 和 LoongArch，其中 x86 使用 CISC 指令集，后三者使用 RISC 指令集。
- 中间语言分为后缀式、图表示法和三地址码三种形式。
- 编译器由词法分析器、语法分析器、语义分析与中间代码生成器、代码优化器、目标代码生成器共 5 个阶段性处理器组成，穿插符号表管理、错误处理两个过程。
- 中间代码优化之前和之后的部分分别称为编译器前端和后端。

山东大学
SHANDONG UNIVERSITY

第一章 编译器概述

The End

谢谢

授 课 教 师 ：郑艳伟
手 机 ：18614002860（微信同号）
邮 箱 ：zhengyw@sdu.edu.cn