

张青林 主编

汇编语言 程序设计

UIBIAN YUYAN CHENGXU SHEJI

上海科学技术出版社

(续表)

功能	入口参数	出口参数
设备状态标志		设备状态标志
有效		有效
无效		无效
设备类型		设备类型
存储的参数值		存储的参数值
本地设备名地址		本地设备名地址
远程名地址		远程名地址
		AX 为错误
		ES: DI = 目标地址
		ES: DI = 目标地址
		地址
		地址
		地址

汇编语言程序设计

张青林 主编

图书在版编目(CIP)数据
 张青林主编. 汇编语言程序设计. 上海: 上海科学技术出版社, 2008.
 ISBN 978-7-313-02323-1
 I. 张... II. 张... III. 汇编语言-程序设计 IV. TP739
 中国版本图书馆CIP数据核字(2007)第 203323 号

上海科学技术出版社

2008年2月第1次印刷
 元: 28.00

内 容 提 要

本书首先介绍汇编语言概念及与汇编语言课程相关的基础知识。以计算机系统组成为线索,阐述了 8086/8088 CPU 的结构,进而介绍其内部寄存器名称和基本作用,并对存储器的组织以及分段情况作了分析讲解。接着,介绍 Intel 8086/8088 的寻址方式、指令系统和常用伪指令。以 MASM 为汇编器,讲解了汇编语言程序的上机过程。为了避免学生集中学习指令而感到知识容量大、密集度高、枯燥乏味的缺憾,本书将部分指令同顺序程序、分支程序、循环程序、子程序设计结合起来讲解。上机实验指导部分先从寄存器、存储器操作入手,到 DEBUG 环境下编程,再过渡到 MASM 汇编器编程。增加了基础操作的内容,丰富了实践教学。为适合不同层次人员的学习,本书将 80x86/Pentium 部分指令及编程技术、汇编语言程序综合举例列入选学内容。

本书可作为高职高专、成人高校计算机类、电子类、电气类、自动化类等各专业的教材,也可作为专业技术人员、自学者的参考书。

图书在版编目(CIP)数据

汇编语言程序设计 / 张青林主编. — 上海: 上海科学技术出版社, 2008.2

ISBN 978 - 7 - 5323 - 9271 - 1

I. 汇... II. 张... III. 汇编语言 - 程序设计 IV. TP313

中国版本图书馆 CIP 数据核字 (2007) 第 202337 号

上海世纪出版股份有限公司
上海科学技术出版社 出版、发行

(上海钦州南路 71 号 邮政编码 200235)

常熟市兴达印刷有限公司印刷

新华书店上海发行所经销

开本 787 × 1092 1/16 印张 15.25

2008 年 2 月第 1 版

2008 年 2 月第 1 次印刷

定价: 28.00 元

如发生质量问题,读者可向工厂联系调换

前言

当今时代,计算机技术已成为每个人必须掌握的文化基础。计算机科学家们(或专家们)为了使用的方便和普及的需要,为广大使用者构建了由“窗口+图标+按钮+自然语言表述”的界面。它给人们带来方便的同时,也让“界面”遮挡了人们的视野。如果要进一步用好、用活计算机,开掘它潜在的软、硬件资源,就必须透过这一“界面”,向更深层次进发,提升计算机知识的深度和广度。“汇编语言程序设计”就是能拔高计算机知识层次,扩大专业相关知识面的一门重要课程。它是大专院校计算机类、电子类、电气类、自动化类专业必修的核心课程之一,是“微机原理及应用”、“微机接口技术”、“操作系统”等其他主干专业课的先修课。本课程对于训练学生理解信息处理过程和原理,掌握程序设计技术,熟悉上机及程序调试技术都有着重要作用。而且,该课程还是国家计算机等级考试PC技术课程之一。

本书以Intel 8086/8088为基本机型,详细介绍了8086/8088微处理器的寻址方式和指令系统,结合程序设计的需要,介绍了常用伪指令;以顺序程序、分支程序、循环程序和子程序设计为引领,详细讲解了编制汇编语言程序的要领、方法和技巧。在本书的最后部分,介绍了80x86/Pentium部分新增指令和80x86的编程技术。为适应不同基础学生的需求,兼顾提高学习兴趣的考虑,增加了汇编语言程序设计的综合举例和练习,供学有余力的学生选学。

在长期的教学实践中,我们深深感到,由于“汇编语言程序设计”这门课程自身的特点,学生学习有难度,普遍觉得理解记忆的分量较重而且对理论教学与上机操作的衔接不满意。如果编制源程序后,用汇编程序(或称汇编器)汇编,从教学进程上讲,必须讲完指令和伪指令之后,才能介绍汇编语言程序上机过程,从而导致理论教学与实践教学脱节,即上机操作在时间上后延较长,一般要到后半学期才能够上机,这无论是对教还是对学都是不利的。针对这一点,我们在教材的内容遴选,实践教学安排上作了有益的尝试和改革,充分体现以人为本,尊重学生对教学的合理要求,力求达到以下特点:

1. 选择合适的教学基点,取材适宜,内容精练,突出重点,化解难点。比如,将“指令系统”化整为零:即先介绍常用基本指令,形成基本的知识脉络。

而把转移指令、循环指令及子程序调用与返回指令,插入到相关程序设计章节中去讲解,旨在消除学生连续学习指令系统,造成知识容量大,纯理论教学的那种单调乏味的感觉。

2. 上机操作内容分为三个方面:

(1) 在启动 DEBUG 之后,做寄存器、存储器操作。

(2) 在 DEBUG 环境下做基本指令的操作、简单程序的运行、简单问题的程序编制与调试。

(3) 利用汇编器进行程序的编制、程序的上机调试运行。这是我们在已有的实践教学基础上作出的安排,虽然 DEBUG 下操作不是我们教学的立足点,但能让学生尽早上机操作,及时验证、运用和巩固所学知识。实践证明,这样做与教学的衔接很紧密,教学效果明显改善。

3. 大量的程序设计举例基本上都配有程序流程图,旨在将理论叙述与形象直观图示相结合,便于学生理解接受。

书中打“*”的章节为选学内容。

在教学具体安排上,编者建议用 48 学时讲授基本内容(第 1~第 8 章),讲完第 2 章即能安排学生上机操作,上机可安排 24 学时。如有更多学时,可选讲第 9、第 10 章,或安排学生自学。

本书可作为高等专科学校、职业技术学院、成人高等院校、本科院校举办的职业技术学院计算机、网络技术、信息管理、应用电子、电气技术、测控技术与仪器等专业的教材,也可供其他专业学生和有关专业技术人员参考,或作为自学用书。

本书由安徽经济管理学院张青林任主编,李明才、何强、杨振宇任副主编,安徽水利水电职业学院汪永华任主审。书中第 1、第 2 章和附录 1 和 2 由安徽水利水电职业学院何强编写;第 3、第 9 章由安徽职业技术学院李明才编写;第 4、第 10 章和上机实验指导由安徽交通职业技术学院杨振宇编写;第 5、第 6、第 7、第 8 章及上机实验指导的部分内容由张青林编写,并由张青林对全书进行了统稿。汪永华对全书作了仔细的审阅和修改,在此深表感谢!

由于编者水平所限,书中错误、不妥之处在所难免,恳请各位专家和读者提出宝贵意见。

编者

目录

CONTENTS

第1章 汇编语言及相关基础知识 /1

- 1.1 汇编语言概述 /1
 - 1.1.1 汇编语言的概念 /1
 - 1.1.2 汇编语言的特点 /2
 - 1.1.3 汇编语言的应用场合 /3
- 1.2 常用数制及其转换 /3
 - 1.2.1 常用数制 /3
 - 1.2.2 数制的转换 /4
- 1.3 无符号数和有符号数 /5
- 1.4 原码、反码和补码 /6
- 1.5 BCD码和ASCII码 /8
 - 1.5.1 BCD码 /8
 - 1.5.2 字符编码(ASCII码) /9
- 1.6 基本逻辑运算 /10
- 习题 /11

第2章 微型计算机内部结构及编程模型 /13

- 2.1 计算机系统组成 /13
 - 2.1.1 硬件系统 /14
 - 2.1.2 软件系统 /15
- 2.2 8086/8088 CPU组成 /16
 - 2.2.1 8086/8088 CPU的基本组成 /16
 - 2.2.2 8086/8088 寄存器组 /18
- 2.3 8086/8088的存储器组织 /22
 - 2.3.1 存储单元的地址和内容 /22
 - 2.3.2 存储器的分段 /22

- 2.3.3 物理地址的形成 /23
- 2.3.4 堆栈 /23
- 2.4 DEBUG的使用 /24
 - 2.4.1 DEBUG程序使用 /24
 - 2.4.2 DEBUG的常用命令 /24
- 习题 /28

第3章 8086/8088的寻址方式和基本指令 /30

- 3.1 概述 /30
- 3.2 与数据有关的寻址方式 /31
 - 3.2.1 立即寻址方式 /31
 - 3.2.2 寄存器寻址方式 /31
 - 3.2.3 直接寻址方式 /31
 - 3.2.4 寄存器间接寻址方式 /32
 - 3.2.5 寄存器相对寻址方式 /32
 - 3.2.6 基址变址寻址方式 /33
 - 3.2.7 相对基址变址寻址方式 /33
- 3.3 8086/8088基本指令 /34
 - 3.3.1 数据传送指令 /34
 - 3.3.2 堆栈操作指令 /35
 - 3.3.3 标志操作指令 /36
 - 3.3.4 算术运算指令 /36
 - 3.3.5 逻辑运算指令 /41
 - 3.3.6 移位指令 /42
 - 3.3.7 串操作指令 /44

- 3.3.8 中断指令 /45
- 3.3.9 输入/输出指令 /46
- 3.3.10 处理器控制指令 /47

习 题 /48

第 4 章 8086/8088 伪指令及上机过程
/50

- 4.1 汇编语言语句格式 /50
 - 4.1.1 名字项 /50
 - 4.1.2 操作项 /51
 - 4.1.3 操作数项 /51
 - 4.1.4 注释项 /55
 - 4.2 伪指令语句 /55
 - 4.2.1 段定义伪操作 /56
 - 4.2.2 程序的开始和结束伪指令 /57
 - 4.2.3 数据定义伪指令 /58
 - 4.2.4 过程定义伪指令 /59
 - 4.2.5 表达式赋值伪操作 /59
 - 4.3 宏指令 /60
 - 4.4 DOS 功能调用 /61
 - 4.4.1 输入字符功能 /62
 - 4.4.2 输出字符功能 /63
 - 4.4.3 输出字符串功能 /63
 - 4.4.4 输入字符串功能 /64
 - 4.4.5 终止程序功能 /65
 - 4.5 汇编语言程序的上机过程 /65
 - 4.5.1 软件环境 /66
 - 4.5.2 汇编语言源程序上机操作说明 /66
- 习 题 /70

第 5 章 顺序程序设计 /72

- 5.1 汇编语言程序设计概述 /72
 - 5.1.1 汇编语言程序设计的基本步骤 /73
 - 5.1.2 汇编语言程序设计举例 /73
- 5.2 流程图的画法规定 /75

- 5.3 顺序程序设计 /76
 - 5.3.1 顺序程序的结构形式 /76
 - 5.3.2 顺序程序设计 /77

习 题 /81

第 6 章 控制转移指令与分支程序设计
/84

- 6.1 与转移有关的寻址方式 /84
 - 6.1.1 段内直接寻址方式 /85
 - 6.1.2 段内间接寻址方式 /85
 - 6.1.3 段间直接寻址方式 /86
 - 6.1.4 段间间接寻址方式 /87
 - 6.2 控制转移指令 /87
 - 6.2.1 无条件转移指令 /87
 - 6.2.2 条件转移指令 /89
 - 6.3 分支程序设计 /92
 - 6.3.1 分支程序的结构 /92
 - 6.3.2 双分支程序设计 /93
 - 6.3.3 多分支程序设计 /99
- 习 题 /107

第 7 章 循环指令与循环结构程序设计
/110

- 7.1 循环指令 /110
- 7.2 循环程序的结构 /112
 - 7.2.1 概述 /112
 - 7.2.2 循环程序的结构形式 /117
- 7.3 单重循环程序设计 /118
 - 7.3.1 单重循环程序设计方法 /118
 - 7.3.2 单重循环程序设计举例 /120
- 7.4 循环程序的控制方法 /123
 - 7.4.1 计数控制法 /123
 - 7.4.2 条件控制法 /124
 - 7.4.3 逻辑尺控制法 /124
 - 7.4.4 开关控制法 /126
- 7.5 多重循环程序设计 /128

习 题 /130

第 8 章 子程序相关指令与子程序设计
/134

- 8.1 子程序调用与返回指令 /134
 - 8.1.1 子程序调用指令 /134
 - 8.1.2 子程序返回指令 /136
- 8.2 子程序的基本构成 /138
 - 8.2.1 子程序及其调用与返回
/138
 - 8.2.2 子程序说明信息 /138
 - 8.2.3 寄存器的保护与恢复
/139
- 8.3 子程序的参数传递方法 /140
- 8.4 子程序设计 /141
 - 8.4.1 子程序定义及格式要求
/141
 - 8.4.2 子程序设计方法 /142
- 8.5 子程序的嵌套与递归 /148
 - 8.5.1 子程序的嵌套 /148
 - 8.5.2 子程序递归 /148

习 题 /152

*** 第 9 章 80x86/Pentium 部分新增指令**
/155

- 9.1 Intel 系列 CPU 简介 /155
- 9.2 80286 增扩指令 /156
- 9.3 80386 增扩指令 /157
- 9.4 80486 增扩指令 /159
- 9.5 Pentium 增扩指令 /159
- 9.6 80x86 指令集选择伪指令 /160

习 题 /160

*** 第 10 章 程序设计综合示例** /161

- 10.1 算术运算程序设计 /161
 - 10.2 非数值处理程序设计 /164
 - 10.3 发声程序设计 /166
 - 10.4 图形显示程序设计 /171
 - 10.5 动画程序设计 /173
- 习 题 /179

**实验一 显示和修改寄存器及显示存储区
操作** /180

**实验二 显示和修改存储单元内容及编程
操作** /182

**实验三 内存操作数、寻址方法和基本指令
操作** /184

实验四 数据的建立与传送操作 /187

实验五 算术、逻辑、移位及串指令的操作
/189

实验六 数据串传送和查表程序 /191

实验七 汇编语言程序的上机过程操作
/193

实验八 分支程序、顺序程序设计 /198

实验九 循环程序设计 /200

实验十 子程序设计 /202

实验十一 统计学生成绩程序设计 /205

实验十二 学生成绩名次表程序设计 /207

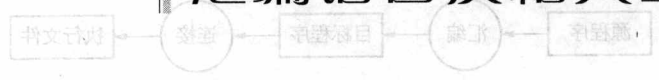
附录 1 80x86 指令表 /211

附录 2 DOS 系统功能调用 (INT 21H)
/229

参考文献 /236

第 1 章

汇编语言及相关基础知识



汇编语言是面向计算机的指令系统，其指令与机器指令是一一对应的。

汇编语言是面向计算机的指令系统，其指令与机器指令是一一对应的。汇编语言是面向计算机的指令系统，其指令与机器指令是一一对应的。

>> 本章学习要点:

- (1) 掌握有关汇编语言的基本概念,包括:程序、指令的概念,汇编过程及汇编语言的特点、应用场合等。
- (2) 掌握常用数制及其转换,包括:二进制、十进制、八进制、十六进制之间的转换,以及 8421BCD 码和 ASCII 码。
- (3) 理解计算机中数的表示方法,以及有关带符号数、无符号数、原码、补码、反码和运算中的溢出问题。
- (4) 掌握基本的逻辑运算,包括:与运算、或运算、非运算和异或运算。

1.1 汇编语言概述

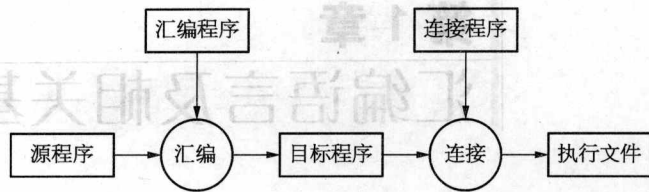
1.1.1 汇编语言的概念

汇编语言是一种面向机器的语言,其指令与机器指令是一一对应的。汇编语言用符号和文字来表示指令,所以它又称为符号语言。

所谓指令是指控制计算机执行某一特定操作的命令。一台计算机所能识别的指令的全体称为指令系统,它反映了计算机基本功能的强弱。

而机器指令是以二进制代码形式表示的、能直接为计算机识别并执行的命令,它通常由操作码和操作数两部分组成。8086/8088 的指令系统有 100 多条基本指令。

程序是按照确定的算法解决具体问题所必需的指令序列,由数据、指令和字符等构成,在程序执行前应预先将其以二进制代码的形式存储在存储单元中。用汇编语言编写的程序是不能被计算机直接识别和执行的(同用高级语言编写的程序一样),它需要翻译成目标程序后方可执行,这个过程我们称为汇编(如“由汇编语言源程序到执行文件的处理过程图”所示)。汇编语言的使用虽然不如高级语言简单方便,但因它与机器语言是一一对应的,故可充分利用计算机硬件系统的特性,提高编程技巧和编程质量。另外,利用汇编语言处理 I/O 设备是汇编语言的独到之处,所以它是无法被其他语言所取代的。汇编程序的类型有:自汇编程序、交叉汇编程序、微汇编程序、浮动汇编程序和宏汇编程序。



由汇编语言源程序到执行文件的处理过程图

汇编语言(ASM)虽然较机器语言在阅读、记忆及编写方面都提高了很多,但对描述任务、编程设计而言仍不方便,于是产生了具有机器语言优点,又能较好地面向问题的语言,即宏汇编语言(MASM)。

宏汇编语言不仅包含一般汇编语言的功能,而且用了高级语言使用的数据结构,是一种接近高级语言的汇编语言。例如,它提供了记录、结构和字符串操作;具有宏处理、条件汇编及磁盘操作系统 DOS 功能调用等多种功能;程序的开发以及调试手段也比较完善,因此,宏汇编语言是一种更高级的汇编语言。

1.1.2 汇编语言的特点

汇编语言相对机器语言而言易记好用,但远不如高级语言方便、实用,而且编写同样的程序,使用汇编语言比使用高级语言花费的时间更多,调试和维护更困难。既然如此,为什么还要使用汇编语言呢?主要有两个原因,即语言的性能和对计算机的完全控制能力。汇编语言具有如下五个特点:

1. 执行速度快

汇编语言程序比高级语言程序执行得更快,而程序的执行速度对于某些应用来说是至关重要的。对于这些应用,单纯使用高级语言往往达不到速度要求,单纯使用汇编语言编写程序也不是最好的方案,所以,许多成功的大型应用程序往往使用混合编程。首先使用高级语言编写整个程序,然后测试程序的执行时间,再使用汇编语言重写其中最费时间的部分。这样做的依据是,在实际使用中,程序的大部分执行时间通常都花费在一小部分代码上。

2. 程序体积小

汇编语言程序的体积比高级语言程序更小。在某些情况下,设备中的嵌入式处理器往往只有很少的内存,例如,智能卡中有 CPU,但是智能卡中很难有 1 MB 以上的内存,也不可能带分页的硬盘,而智能卡又必须执行复杂的加密解密计算,使用汇编语言可能是惟一的方法。个人数字助理(PDA)和其他使用电池作为能源的无线电子设备,为了节省电池的电力,往往也只有很少的内存,它们也需要使用短小精悍、高效率的机器代码。

3. 可以直接控制硬件

某些应用程序要求能够完全控制计算机硬件,这也必须使用汇编语言。如操作系统中的低级中断和陷阱处理程序,以及许多嵌入式实时系统中的设备控制程序等,都属于这一类应用。

4. 可以方便地编译

编译器可以产生供编程者使用的汇编程序,或者自己执行汇编过程。因此,为了解编译器的工作原理,必须首先理解汇编语言。

5. 辅助计算机工作者掌握计算机体系结构

研究汇编语言可以使人们掌握计算机的实际结构,特别是对于学习计算机体系结构的人员,编写汇编语言是在结构层理解计算机的惟一途径。

1.1.3 汇编语言的应用场合

随着软件技术的发展,汇编语言的应用领域越来越小,但是它的特点又决定了它的必然作用。下面罗列了汇编语言的部分应用场合:

(1) 程序要具有较快的执行时间,或者只能占用较小的存储容量。例如,操作系统的核心程序段,实时控制系统的软件,智能仪器仪表的控制程序等。

(2) 程序与计算机硬件密切相关,程序要直接、有效地控制硬件。例如,I/O 接口电路的初始化程序段,外部设备的低层驱动程序等。

(3) 大型软件需要提高性能、优化处理的部分。例如,计算机系统频繁调用的子程序和动态链接库等。

(4) 没有合适的高级语言或只能采用汇编语言的时候。例如,开发最新的处理器程序时,暂时没有支持新指令的编译程序等。

汇编语言还有许多其他的实际应用,例如,分析具体系统尤其是该系统的低层软件、加密解密软件、分析和防治计算机病毒等。

1.2 常用数制及其转换

1.2.1 常用数制

数制是数的表示及计算方法。数制型数据是有大小的,人们习惯使用的是十进制数,但在计算机内,各种信息都是用二进制代码形式表示的。为了书写方便,二进制数也常用八进制数和十六进制数表示。但任何一种数制都具有以下几个要点:

(1) 基数: 每种进制的基数就是该进制本身,如二进制的基数是 2。

(2) 数码: 二进制的数码为 0, 1; 十进制的数码为 0~9; 八进制的数码为 0~7; 十六进制的数码为 0~9、A~F 等。

(3) 进位原则: 十进制逢十进一,二进制逢二进一等。

(4) 位权: 指每一位数位上数码所具有的权,如十进制的位权为 10^i ,二进制的位权为 2^i 等, i 的取值为整数。

四种常用进制数的关系如表 1-1 所示。

表 1-1 四种常用进制数的对照表

十进制	二进制	八进制	十六进制	十进制	二进制	八进制	十六进制
0	0000	0	0	8	1000	10	8
1	0001	1	1	9	1001	11	9
2	0010	2	2	10	1010	12	A
3	0011	3	3	11	1011	13	B
4	0100	4	4	12	1100	14	C
5	0101	5	5	13	1101	15	D
6	0110	6	6	14	1110	16	E
7	0111	7	7	15	1111	17	F

由于不同位置的权值不同,因此同一数码在不同位置上,表示的值也是不同的。每个数位上的值等于该位置上的数码与该位置的权值的乘积,相邻数位中,高位权与低位权之比即是该进制的基数。利用下式,可将任意进制的数转化为十进制数,其中, a 表示某数制的数码, R 表示基数(底), R^i 表示数位的权, m 和 n 为正整数。

$$N = \sum_{i=-m}^{n-1} a_i R^i = a_{n-1} R^{n-1} + a_{n-2} R^{n-2} + \dots + a_1 R^1 + a_0 R^0 + \dots + a_{-m} R^{-m}$$

1.2.2 数制的转换

下面具体介绍各种数制之间的转换方法。

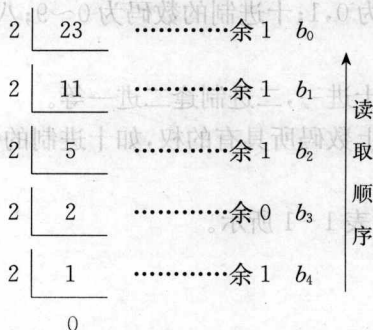
1. 将十进制数转换成二进制数、八进制数或十六进制数

转换方法: 整数部分除以基数(2、8 或 16)取余数,小数部分乘以基数(2、8 或 16)取整数。

【例 1.1】 试将十进制数 23.125 转换成二进制数。

解: (1) 整数部分的转换: 除 2 取余。

根据“除 2 取余”法的原理,按如下步骤转换:



例：(23)₁₀=(10111)₂

(2) 小数部分的转换：乘2取整。

小数部分的转换正好与整数部分的转换相反，先将0.125乘以2得结果0.25，则在右边记下整数位0，再将去整数位后的0.25乘以2，记下结果的整数位，依此类推，直到最后小数部分为0或结果已达到精度要求为止。最后把刚记下的整数位顺着排列即得结果，用“乘2取整”法，按如下步骤转换：

$$\begin{aligned} 0.125 \times 2 &= 0.25 && \dots\dots\dots 0 && b_{-1} \\ 0.25 \times 2 &= 0.5 && \dots\dots\dots 0 && b_{-2} \\ 0.5 \times 2 &= 1 && \dots\dots\dots 1 && b_{-3} \end{aligned}$$

因此：(0.125)₁₀=(0.001)₂

所以，十进制数23.125转换成二进制数为10111.001。十进制数转换成八进制数或十六进制数的方法同上，(45.25)₁₀=(55.2)₈=(37.4)₁₆。

2. 将二进制数、八进制数或十六进制数转换成十进制数

转换方法：按位权展开求和。

【例 1.2】 $(10111)_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = (23)_{10}$
 $(572.34)_8 = 5 \times 8^2 + 7 \times 8^1 + 2 \times 8^0 + 3 \times 8^{-1} + 4 \times 8^{-2} = (378.4375)_{10}$
 $(35A)_{16} = 3 \times 16^2 + 5 \times 16^1 + 10 \times 16^0 = (858)_{10}$

3. 二进制数与八进制数、十六进制数的转换

因为2³=8，所以二进制数转换成八进制数只需将二进制数从小数点开始每3位转换成1位八进制数(整数由左向右，小数相反，不足3位的用0补齐)。

【例 1.3】 $(101\ 111\ 010.\ 011\ 100)_2 = (572.34)_8$

八进制数转换成二进制数只需将每1位八进制数用3位二进制数表示，小数点位置不变。

【例 1.4】 $(175.54)_8 = (001\ 111\ 101.\ 101\ 100)_2 = (1111101.1011)_2$

十六进制数与二进制数的转换只需将上面二进制数与八进制数转换方法中每隔3位改为每隔4位即可。

【例 1.5】 $(9B28)_{16} = (1001\ 1011\ 0010\ 1000)_2 = (1001101100101000)_2$

1.3 无符号数和有符号数

各种数据在计算机中的表示形式称为机器数，其特点是数的符号用“0”和“1”表示，其中，“0”表示正号，“1”表示负号，小数点隐含表示而不占位置。机器数对应的实际数值称为该机器数的真值。

机器数分为无符号数和有符号数两种。无符号数表示正数，在机器数中没有符号位。对于无符号数，若约定小数点的位置在机器数的最低位之后，则是纯整数；若约定小数点的位置在机器数的最高位之前，则是纯小数。对于有符号数，机器数的最高位是表示正、负的符号位，其余二进制位表示数值。若约定小数点的位置在机器数的最低位之后，则是纯整

数;若约定小数点的位置在机器数的最高位之前(符号位之后),则是纯小数。这种表示数的方式称为定点数。

定点数表示法的缺点在于其形式过于僵硬,固定的小数点位置决定了固定位数的整数部分和小数部分,不利于同时表示特别大的数或者特别小的数,所以绝大多数现代的计算机系统采用了所谓的浮点数表示方式。这种表示方式利用科学计数法来表示实数,即用一个尾数(M)、一个基数(B)、一个指数(E)和一个表示正负的符号来表示实数,直观表示为 $N = M \times B^E$ 。如 123.45 用十进制科学计数法可以表示为 1.2345×10^2 ,其中 1.2345 为有效数字(即尾数),10 为基数,2 为指数。浮点数利用指数达到了浮动小数点的效果,从而可以灵活地表示更大范围的实数。

1.4 原码、反码和补码

为了运算方便,机器数有不同的编码方法,称为码制。常用的码制有原码、反码、补码和移码等。

1. 原码

原码又称为符号绝对值码。数据最高位为符号位,正数用“0”表示,负数用“1”表示。其他位为数据位,用二进制数的绝对值表示。原码与真值转换方便,但做加减运算不方便,而且零有“+0”和“-0”两种表示方法。

【例 1.6】当机器字长为 8 位时:

$[+1]_{原} = 00000001B$	$[-1]_{原} = 10000001B$
$[+127]_{原} = 011111111B$	$[-127]_{原} = 111111111B$
$[+0]_{原} = 00000000B$	$[-0]_{原} = 10000000B$

2. 反码

正数的反码表示与原码相同;负数的反码,符号位为“1”不变,数值位由其绝对值各位取反得到。零的反码也有“+0”和“-0”两种表示方法。反码因运算不便而较少使用。

【例 1.7】当机器字长为 8 位时:

$[+1]_{反} = 00000001B$	$[-1]_{反} = 11111110B$
$[+127]_{反} = 011111111B$	$[-127]_{反} = 10000000B$
$[+0]_{反} = 00000000B$	$[-0]_{反} = 111111111B$

3. 补码

为了加减运算的方便引入了补码概念,其关键思想是用加法代替减法。正数的补码与原码表示相同;负数的补码,符号位用“1”表示,数值位用其绝对值的补码表示(即原码各位求反,末位加 1)。在补码表示中,0 有唯一的编码: $[+0]_{补} = [-0]_{补} = 00000000B$ 。

【例 1.8】当机器字长为 8 位时:

$[+1]_{补} = 00000001B$	$[-1]_{补} = 111111111B$
$[+127]_{补} = 011111111B$	$[-127]_{补} = 10000001B$

补码的表示范围比原码和反码略宽,在定点小数中,补码可以表示 -1,即 $[-1]_{补} =$

10000B, 而原码和反码不能表示绝对值为 1 的数。

【例 1.9】 假定机器字长为 8 位, 试写出 122 的原码、反码和补码。

$$[122]_{\text{原}} = [122]_{\text{反}} = [122]_{\text{补}} = 01111010\text{B}$$

【例 1.10】 假定机器字长为 8 位, 试写出 -45 的原码、反码和补码。

$$[-45]_{\text{原}} = 10101101\text{B}$$

$$[-45]_{\text{反}} = 11010010\text{B}$$

$$[-45]_{\text{补}} = 11010011\text{B}$$

对于用补码表示的负数, 首先认定它是负数, 然后用求它的补码的方法得到它的绝对值, 即可求得该负数的值。例如, 补码数 11110011B 是一个负数, 求该数的补码为 00001101B, 该数相应的十进制数为 13, 故求出 11110011B 为 -13D。

4. 溢出

各种数据编码都有其数据表示范围, 如果在运算过程中出现的数据超出编码表示的范围, 称为溢出。对于加法, 只有在正数加正数和负数加负数两种情况下才会产生溢出, 即符号相同的两个数相加可能会产生溢出, 而符号不同的两个数相加不会产生溢出。对于减法, 只有在正数减负数和负数减正数两种情况下才会产生溢出, 即符号不同的两个数相减可能会产生溢出, 而符号相同的两个数相减不会产生溢出。

在微处理器中, 使用补码进行运算是十分方便的, 它使同一个微处理器中既能运算带符号数又能运算不带符号的数。而在进行带符号数的加减运算时, 应把参与运算的数据转换成补码形式进行运算。当使用 8 位二进制数表示带符号的数时, 它所能表示的数值范围在 $(-128)_{10} \sim (+127)_{10}$ 之间, 如果相加结果超出了这个范围, 就会导致错误发生。

$$[X + Y]_{\text{补}} = [X]_{\text{补}} + [Y]_{\text{补}}$$

$$[X - Y]_{\text{补}} = [X]_{\text{补}} + [-Y]_{\text{补}}$$

【例 1.11】 两个带符号的数 01000001B 与 01000011B 相加:

01000001	+	01000011	0000
		10000100	0001

本例中是两个正数相加, 但结果却是一个负数(符号位为 1)。显然, 这个结果是错误的, 出现这种错误的原因就在于这两个数相加的结果超过了 8 位二进制带符号数所能表示的数值范围。

【例 1.12】 两个带符号的负数 10001000B 和 11101110B 相加:

10001000	+	11101110	0110
		01110110	0001

由于规定用 8 位二进制数来表示带符号的数, 故忽略作为进位位的第九位。按 8 位二进制数来解释这两个带符号的负数的相加, 其结果为一个正数。很明显, 结果是错误的。

【例 1.13】 两个无符号数 11111101B 和 00000011B 相加:

$$\begin{array}{r}
 11111101 \\
 + 00000011 \\
 \hline
 1 \quad 00000000
 \end{array}$$

从相加计算的结果来看,如果微处理器只有 8 位,也就是用 8 位二进制数来解释运算的结果,则将出现错误。因此,在微处理器中设有专门的一位,称为进位位,它将用于保存第九位,以防丢失信息。

1.5 BCD 码和 ASCII 码

1.5.1 BCD 码

计算机内部使用的是二进制数,但二进制数书写冗长,阅读不便,所以,在输入输出时人们仍习惯使用十进制数。如果计算量不大,可采用二进制数对每一位十进制数字进行编码的方法来表示一个十进制数,这种数叫做 BCD(Binary-Coded Decimal)码。由于在机器内采用 BCD 码进行运算,绕过了二进制、十进制间的复杂转换环节,从而节省了机器时间。

BCD 码有多种形式,最常用的是 8421BCD 码,它是用 4 位二进制数对十进制数的每一位进行编码,这 4 位二进制码的值就是被编码的一位十进制数的值。由于 4 位二进制数可以表示 16 种状态,所以丢弃最后 6 种状态,选用 0000~1001 来表示 0~9 十个数值。这种编码又叫做 8421 码,如表 1-2 所示。

表 1-2 十进制数与 8421BCD 码的对应关系

十进制数	BCD 码	十进制数	BCD 码
0	0000	10	00010000
1	0001	11	00010001
2	0010	12	00010010
3	0011	13	00010011
4	0100	14	00010100
5	0101	15	00010101
6	0110	16	00010110
7	0111	17	00010111
8	1000	18	00011000
9	1001	19	00011001

【例 1.14】 将十进制数 69.25 转换成 BCD 码。

(6) 9 . 2 5

0110 1001. 0010 0101

结果为 69.25=(01101001.00100101)BCD

【例 1.15】 将 BCD 码 100101111000.01010110 转换成十进制数。

1001 0111 1000. 0101 0110

× 9 7 8. 5 6

结果为(100101111000.01010110)BCD=978.56

1.5.2 字符编码(ASCII 码)

鉴于信息交换的重要性以及统一文字符号的编码标准,让不同厂家不同机型的计算机都能使用同一套标准化的信息交换码,于是美国国家标准局特别制定了 ASCII 码(America Standard Code for Information Interchange,美国信息交换标准码),作为数据传输的标准码。

ASCII 码的每个字符用 7 位二进制数表示,其排列次序为 $d_6d_5d_4d_3d_2d_1d_0$, d_6 为高位, d_0 为低位。而一个字符在计算机内实际是用 8 位表示。正常情况下,最高一位 d_7 为 0。7 位二进制数共有 128 种编码组合,可表示 128 个字符,其中数字 10 个、大小写英文字母共 52 个、其他字符 32 个和控制字符 34 个。

(1) 数字 0~9 的 ASCII 码为 30H~39H。

(2) 大写英文字母 A~Z 的 ASCII 码为 41H~5AH。

(3) 小写英文字母 a~z 的 ASCII 码为 61H~7AH。

对于 ASCII 码表中的 0、A、a 的 ASCII 码 30H、41H、61H 应尽量记住,其余的数字和字母的 ASCII 码可按数字和字母的顺序以十六进制的规律写出,具体的 ASCII 码见表 1-3。

表 1-3 ASCII 码表

$d_6d_5d_4$ \ $d_3d_2d_1d_0$		0		1		2		3		4		5		6		7	
		000		001		010		011		100		101		110		111	
0	0000	NUL	DLE	SP	0	@	P	,	p								
1	0001	SOH	DC1	!	1	A	Q	a	q								
2	0010	STX	DC2	"	2	B	R	b	r								
3	0011	ETX	DC3	#	3	C	S	c	s								
4	0100	EOT	DC4	\$	4	D	T	d	t								
5	0101	ENQ	NAK	%	5	E	U	e	u								
6	0110	ACK	SYN	&	6	F	V	f	v								