

C 程序设计基础教程

主编 陈章进

上海大学出版社

· 上海 ·

内 容 提 要

本书将 C 语言作为“第一计算机语言”，兼顾一般程序设计和 C 语言程序之间共性与个性关系。

全书共分三大部分十个章节，第一部分为第一、二章，详细阐述二进制和各方面基础，使用最常用的数据类型、运算符和语句，分析解题过程中的各个步骤，为学习编程打下坚实的基础；第二部分为第三至第六章，分别讲述一般编程中的主要组成部分，包括数据类型、控制语句、数组和函数；第三部分突出 C 语言特有的内容，包括指针、结构体、文件和预处理等。

本书可作为大学生非计算机专业的程序设计教材，也可供计算机专业学生参考，本书还可供自学使用。

图书在版编目(CIP)数据

C 程序设计基础教程/陈章进主编. —上海: 上海大学出版社, 2005. 9

ISBN 7-81058-904-0

I. C... II. 陈... III. C 语言—程序设计—教材
IV. TP312

中国版本图书馆 CIP 数据核字(2005)第 085245 号

责任编辑 张 苏 江振新 夏 鸣
封面设计 谷夫平面设计工作室

C 程序设计基础教程

主编 陈章进

上海大学出版社出版发行

(上海市上大路 99 号 邮政编码 200444)

(<http://www.shangdapress.com> 发行热线 66135110)

出版人: 姚铁军

*

上海华业装潢印刷厂印刷 各地新华书店经销

开本 787×1092 1/16 印张 19.5 字数 450 千字

2005 年 9 月第 1 版 2005 年 9 月第 1 次印刷

印数: 1-3 100 册

ISBN 7-81058-904-0/TP·034 定价: 30.80 元

C 程序设计基础教程

编委会名单

主 编 陈章进

撰稿人 (按撰写章节先后为序)

陈章进 邹启明 宋兰华

侯庆祥 陈 灏

前 言

随着计算机文化课的推广,计算机操作方面的能力普遍提高,相应地,计算机设计方面的重要性越来越突出,程序设计是计算机软件中最基本的设计课程,C语言是计算机软件语言中使用最广泛、功能最强的语言之一,它适用于编写一般应用程序,也能用来编写计算机系统软件,还适用于嵌入式系统中的程序编写,可以说它是个全方位、多用途、高效率的软件语言。

本书定位是将C语言作为大学“第一计算机语言”,即C语言紧跟在“计算机文化和应用技术”课程后开设,因此本课程肩负着程序设计和C语言两方面的要求,本书充分考虑了程序设计与C语言之间共性与个性的关系。

本着循序渐进、层层提高的原则,本书将学习程序设计归结为“学程三转”,即将全书教学分成三个阶段,每一阶段都对程序设计有个相对完整的阐述,并逐步深入。

第一阶段为第一章与第二章,构建程序设计的基础。它从计算机最基本的二进制和指令运行方式开始,使用最常用和最基本的数据类型、运算符和语句,详细分析程序设计的整个设计过程,包括问题的分析、算法描述、程序结构、正确性分析比较,以及上机过程与调试方法等,是程序设计的基础部分,也是程序设计中最重要的手段。

第二阶段为第三至第六章,逐个展开程序设计中的主要组成部分。在第一阶段的基础上,讲述一般程序设计语言共同的语言特征,包括数据类型、控制语句、数组和函数等,这是程序设计语言中的共性内容。

第三阶段为第七至第十章,深入C语言的各个重要特征。从分析前两阶段各变量的内存安排与使用来阐述指针,并在指针基础上讲述结构体和文件,最后为预处理与位运算等。

本书内容遵循ISO/ANSI标准,并以PC机上的Turbo C 2.0为编程环境,全部例题都在PC机上调试通过。

本书编者为陈章进(第一、二、七章),邹启明(第三、十章和附录)、宋兰华(第四、八章)、侯庆祥(第五章)、陈灏(第六、九章),由陈章进和侯庆祥统稿。范燮昌老师通读全稿,并提出很多宝贵意见。感谢焦政、陆铭、谢建华和卜家岐等老师对本书的支持。

由于作者水平有限,经验不多,本书会有不少缺点或错误,敬请专家和读者指正。

编 者

2005年7月于上海大学

目 录

第一章 C 程序设计基础	1
1.1 二进制基础	1
1.1.1 为什么采用二进制	1
1.1.2 二进制、十六进制和八进制	2
1.1.3 原码、反码与补码	5
1.1.4 模 2^{16} 原则	8
1.1.5 浮点小数的二进制表示	9
1.2 程序员眼中的计算机	11
1.2.1 数据在计算机中的存储形式	11
1.2.2 计算机的运行方式与特点	13
1.3 程序设计的方法	14
1.3.1 程序设计的两大要素	14
1.3.2 算法的描述	16
1.3.3 流程的跟踪执行	19
1.4 C 程序初步	21
1.4.1 计算机语言分类	21
1.4.2 C 语言简史及特点	21
1.4.3 C 程序示例及组成	22
1.5 C 程序解题分析	27
1.5.1 详解判断素数程序	27
1.5.2 综合示例	35
1.5.3 C 程序书写风格	37
1.6 编程学习方法总结	40
习题	41
第二章 上机指导与解题分析	43
2.1 上机编程环境	43
2.2 Turbo C 集成开发环境	44
2.2.1 Turbo C 集成开发环境介绍	44
2.2.2 文本编辑	46
2.2.3 编译和连接	48
2.2.4 程序运行与调试	50
2.2.5 Turbo C 环境设置	54

2.3	常见的变量类型、运算符及输入输出格式	55
2.4	程序举例	57
2.5	Turbo C 上机实验	64
	习题	70
第三章 基本数据类型、运算符与输入输出函数		72
3.1	基本数据类型	72
3.2	常量	73
3.2.1	整型常量	73
3.2.2	实型常量	73
3.2.3	字符常量	73
3.2.4	字符串常量	74
3.2.5	符号常量	75
3.3	变量	75
3.3.1	C 标识符	75
3.3.2	整型变量	76
3.3.3	实型变量	76
3.3.4	字符型变量	77
3.3.5	变量的初始化	77
3.4	C 语言的运算符和表达式	77
3.4.1	算术运算符和算术表达式	78
3.4.2	赋值运算符和赋值表达式	79
3.4.3	关系运算符和关系表达式	80
3.4.4	逻辑运算符和逻辑表达式	80
3.4.5	条件运算符和条件表达式	82
3.4.6	逗号运算符和逗号表达式	83
3.4.7	指针运算符	83
3.4.8	sizeof 运算符	84
3.4.9	不同类型数据之间的转换	84
3.4.10	运算符优先级和结合性	85
3.5	输入输出函数	85
3.5.1	格式化输入输出函数	86
3.5.2	字符输入输出函数	89
	习题	90
第四章 语句与程序控制结构		94
4.1	语句总述	94
4.1.1	说明语句、表达式语句和空语句	94

4.1.2 控制语句	95
4.1.3 复合语句	95
4.2 顺序结构	95
4.3 选择结构	96
4.3.1 if 语句	96
4.3.2 switch 语句	100
4.4 循环结构	105
4.4.1 while 语句	105
4.4.2 do-while 语句	107
4.4.3 for 语句	108
4.4.4 break 语句	110
4.4.5 continue 语句	112
4.4.6 goto 语句	113
4.4.7 多重循环	114
4.5 程序举例	117
习题	122
第五章 数组与字符串	123
5.1 一维数组	123
5.1.1 一维数组的定义和引用	123
5.1.2 一维数组初始化	125
5.1.3 程序举例	126
5.2 二维数组	133
5.2.1 二维数组的定义和引用	133
5.2.2 二维数组初始化	135
5.2.3 程序举例	136
5.3 字符串处理	139
5.3.1 字符数组与字符串	139
5.3.2 字符串的输入和输出	140
5.3.3 字符串处理函数	143
5.3.4 程序举例	145
习题	149
第六章 函数	151
6.1 模块化程序设计思想	151
6.2 模块设计举例	151
6.3 函数概念	152
6.3.1 函数定义	152

6.3.2	函数说明和函数调用	153
6.3.3	形参与实参	156
6.3.4	函数返回值	157
6.3.5	传值与传址	158
6.3.6	数组作为函数的参数的例子	159
6.4	函数的嵌套	161
6.5	函数的递归	161
6.6	变量作用域和存储类别	167
6.6.1	局部变量和全局变量	167
6.6.2	变量存储类别	168
6.7	使用库函数	169
6.8	程序举例	172
	习题	176
第七章	指针	177
7.1	指针概念	177
7.2	指针变量的定义和使用	180
7.3	指针与一维数组	182
7.3.1	一维数组的内存安排	182
7.3.2	指向数组的指针运算	183
7.3.3	下标法与指针法表示的等价性	186
7.3.4	程序举例	187
7.4	指针与函数	189
7.4.1	指针作为函数参数	189
7.4.2	指针作为函数返回类型	192
7.4.3	动态内存分配与释放函数	193
7.4.4	函数指针	195
7.5	指针数组、多级指针与指向一维数组的指针	197
7.5.1	指针数组与多级指针	197
7.5.2	二维数组与指向一维数组的指针	199
7.5.3	二维数据的构造举例	201
7.6	指针与字符串	202
7.6.1	字符串的内存安排	202
7.6.2	字符串处理程序举例	203
7.6.3	多字符串的表示与处理	206
7.6.4	程序举例	210
7.6.5	命令行参数	216
	习题	217

第八章 结构体、联合与枚举	219
8.1 概述	219
8.2 结构体类型定义	219
8.3 结构体变量	221
8.3.1 结构体变量的定义	221
8.3.2 结构体变量的引用	222
8.3.3 结构体变量初始化	224
8.4 结构体数组	225
8.4.1 结构体数组的定义	225
8.4.2 结构体数组初始化与引用	226
8.4.3 结构体数组的输入和输出	226
8.5 指向结构体类型数据的指针	227
8.5.1 定义和初始化	228
8.5.2 用结构体指针引用结构体成员	228
8.5.3 指向结构体数组的指针	228
8.6 结构体与函数	230
8.6.1 向函数传递结构信息	230
8.6.2 结构体指针作为函数返回值	233
8.7 单链表	234
8.7.1 概述	234
8.7.2 结点定义	235
8.7.3 建立链表	235
8.7.4 输出链表	243
8.7.5 插入结点	243
8.7.6 删除结点	246
8.7.7 程序举例	248
8.7.8 其他链表结构	251
8.8 联合与枚举	253
8.8.1 联合类型	253
8.8.2 枚举类型	255
8.8.3 复杂类型举例	257
8.9 用 typedef 定义类型	259
习题	260
第九章 文件	261
9.1 文件概述	261
9.2 文件打开和关闭	262
9.3 文本文件的输入与输出	264

9.3.1 读写一个字符的函数	264
9.3.2 字符串读写函数	266
9.3.3 格式化读写函数	267
9.4 二进制文件的输入与输出	270
9.5 其他文件库函数	273
9.5.1 文件的定位函数	273
9.5.2 出错监测函数	275
习题	275
第十章 预处理与位运算	276
10.1 预处理	276
10.1.1 概述	276
10.1.2 宏定义	276
10.1.3 文件包含	279
10.1.4 条件编译	280
10.2 位运算	282
习题	285
附录 I 常用 ASCII 码对照表	287
附录 II C 语言关键字表	289
附录 III C 语言运算符表	290
附录 IV C 常用库函数	291
参考文献	295

第一章 C 程序设计基础

程序员是在要解决的问题领域与计算机的真实运行之间的一个桥梁,而沟通两者的工具是使用 C 语言,所以要让计算机真正实现预期的目标,必须对这三个方面都有相当程度的理解和把握。本章首先从计算机的基础二进制入手,随之从软件角度说明计算机底层指令级的运行方式;接着将问题领域看成是两大要素的组合,从两个方面摸索解决问题的途径,描述算法并画流程图是可行的有效手段;再对 C 程序做初步的介绍,示例最基本的、最常用的和最重要的 C 程序和语句等。然后,结合以上三个方面,通过由浅入深的两个实际例子,详细分析从问题领域到程序代码的各种方法和每一个步骤,本章最后提出代码的书写风格,并对学习编程做一个总的描述。

1.1 二进制基础

我们常用的记数方法为十进制,即逢十进一,使用 0,1,……,9 这 10 个数码,而二进制则是逢二进一,只使用 0 和 1 这 2 个数码。

二进制是计算机最根本的基础,计算机系统的一切最终都可以归结到二进制,因此不管是硬件设计还是软件设计,都要对二进制有深刻的理解。

1.1.1 为什么采用二进制

初学计算机的人往往奇怪为什么计算机要采用二进制形式,而不是采用更常用的十进制形式,这有以下几个原因。

一、成本更低

从硬件角度来说,要表示一个具体的数,总是需要一定的电路器件,而一定的电路器件所能表示的数的范围也总是有限的,与十进制相比,二进制计数方法可以使用更少的器件,却能表示更大的数据范围,有两种模型可以比较十进制与二进制的效率优劣。

1. 纸张模型

假设一个球队的得分范围为 0 到 999 分,使用较原始的纸张方法来表示得分情况,事先要准备 30 张纸,从左到右平均分成三叠,分别代表百位数、十位数和个位数,每叠纸从上到下依次写上 0 到 9,对应十进制每位数的 10 个数码。现在假设要表示数字“58”,左边表示百位数的纸翻到“0”,中间的十位数翻到“5”,右边的个位数翻到“8”,合起来表示数字“058”,即“58”。由于 $1\ 000=10^3$,表示 0 到 999 这 1 000 个数,十进制方式的总代价为 $3 \times 10=30$ 张纸。

那么采用二进制计数方法代价又是多少呢？由于 $1\ 024 = 2^{10}$ ，所以表示 0 到 1 023 这 1 024 个数，需要 10 叠，每叠 2 张纸，总代价为 20 张，与十进制相比，表示同样范围的数代价只有原来的 $2/3$ 。

在纸张模型中，可以证明三进制才是最佳方式，但与二进制相比优势不大。

2. 算珠模型

考虑使用算盘来表示数字，算盘的列数即是数的位数，算珠的颗数即是代价。

使用二进制方式，每位只需 1 颗算珠，算珠拨上表示 1，拨下表示 0，10 位二进制只需 10 颗算珠。

使用十进制方式，每位需要 5 颗算珠，其中 1 颗表示 5，4 颗表示 1，如果表示 7，则拨上一颗表示 5 的珠子和 2 颗表示 1 的珠子，要表示 0 则拨开所有珠子，3 位十进制总共需要 15 颗算珠。

在算珠模型中，二进制是最佳方式。

二、运算更简单

由于二进制的每一位只有 0 和 1 这两种，因此 2 个二进制数之间的加减乘除等运算也非常简单，比如相乘，它不需要“九九乘法表”。

三、硬件上更容易实现

开关的开与关，指示灯的亮与暗，电压的高与低，电流的通过与截止，逻辑上的真与假，存在性的有与无等等，都可以简单地和二进制中的 0 与 1 相对应。而半导体器件的最基本特性就是开关特性，所以二进制最容易实现，也很容易判断某位数为 0 或 1，不容易误判。

1.1.2 二进制、十六进制和八进制

计算机内部数据都是使用二进制形式，而二进制表示数字时往往位数太长，不易书写和记忆，为了表示方便，经常使用与二进制接近的十六进制或八进制来代替二进制，另一方面，人们习惯使用的仍是十进制，所以要能熟练各进制之间的转换及简单运算。

一、二进制转换到十进制

借助幂级数表示形式可以计算各种进制数的值，如十进制数 234 写成以 10 为基的幂级数形式是（十进制百位、十位、个位的权分别为 10^2 、 10^1 、 10^0 ）：

$$(234)_{10} = 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$$

上式中下标 10 表示十进制，二进制数也有对应的幂级数表示形式，如：

$$(1101)_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

上式中下标 2 表示二进制，右式部分按十进制计算，得

$$(1101)_2 = 8 + 4 + 0 + 1 = (13)_{10}$$

使用表 1-1 可以快速地将二进制转换为十进制数，将二进制各位的权值先计算好并从高到低排列，在该列下写出二进制数，与权值一一对应，累加二进制数中位为 1 对应的权值，即得十进制结果，如二进制 1101 中有 3 个 1，分别对应权值 8、4 和 1，相加得十进制

结果为 13, 又如:

$$(1011010)_2 = 64 + 16 + 8 + 2 = (90)_{10}$$

表 1-1 将二进制转换为十进制的简易方法

二进制的权值		128	64	32	16	8	4	2	1	结果
例 1	二进制					1	1	0	1	
	累加					8	4		1	13
例 2	二进制		1	0	1	1	0	1	0	
	累加		64		16	8		2		90

二、十进制转换到二进制

将二进制幂级数形式中的 2 提取出来, 得

$$(13)_{10} = (1101)_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = ((\underline{1}) \times 2 + \underline{1}) \times 2 + \underline{0} \times 2 + \underline{1}$$

其中, 右式中带下画线部分就是十进制对应的二进制值, 使用长除法可以计算出十进制对应的二进制值:

$$13 \div 2 = 6 \text{ 余 } 1, \text{ 余数 } 1 \text{ 为二进制的最低位}$$

$$6 \div 2 = 3 \text{ 余 } 0$$

$$3 \div 2 = 1 \text{ 余 } 1$$

$$1 \div 2 = 0 \text{ 余 } 1, \text{ 余数 } 1 \text{ 为二进制的最高位}$$

按余数部分从低到高列出, 即得二进制值, 即 $(13)_{10} = (1101)_2$

对于数值不太大的十进制数, 可以将其转化二进制权值的累加和, 再按表 1-2, 将其转化为二进制, 如十进制 13, 先将 13 化为 $8 + 4 + 1$, 然后按权值顺序即得上式结果, 又如 $150 = 128 + 16 + 4 + 2 = (10010110)_2$ 。这种方法只有十进制的加减运算, 比较简单, 也不易出错。

表 1-2 将十进制转换为二进制的简易方法

二进制的权值		128	64	32	16	8	4	2	1
13	分解					8	4		1
	二进制	0	0	0	0	1	1	0	1
150	分解	128			16		4	2	
	二进制	1	0	0	1	0	1	1	0

三、二进制与八进制互换

由于 $8 = 2^3$, 每 3 位二进制对应 1 位八进制, 对应关系如表 1-3 所示。

表 1-3 八进制与二进制的对应关系

八进制	0	1	2	3	4	5	6	7
二进制	000	001	010	011	100	101	110	111

将二进制按 3 位 1 组, 根据表 1-3, 很容易转化为八进制, 如:

$$(10101)_2 = (010\ 101)_2 = (25)_8$$

$$(1001110)_2 = (001\ 001\ 110)_2 = (116)_8$$

反过来, 八进制也很容易转化为二进制, 如:

$$(34)_8 = (011\ 100)_2 = (11100)_2$$

$$(507)_8 = (101\ 000\ 111)_2 = (101000111)_2$$

四、二进制与十六进制互换

由于 $16=2^4$, 每 4 位二进制对应 1 位十六进制, 对应关系如表 1-4 所示。

表 1-4 十六进制与二进制对应关系

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

将二进制按 4 位 1 组, 根据表 1-4, 容易转化为十六进制, 如:

$$(10101)_2 = (0001\ 0101)_2 = (15)_{16}$$

$$(1001110)_2 = (0100\ 1110)_2 = (4E)_{16}$$

反过来, 十六进制也很容易转化为二进制, 如:

$$(34)_{16} = (0011\ 0100)_2$$

$$(A5D)_{16} = (1010\ 0101\ 1101)_2$$

计算机一般使用 8 位、16 位或 32 位二进制来表示一个整数, 转化为十六进制, 正好对应 2 位、4 位及 8 位十六进制, 所以, 多数微机中使用十六进制的频率比八进制高, 写二进制数时, 更多的习惯是使用每 4 位 1 组方式。

五、二进制加减运算

二进制只有 0 和 1 两种数字, 直接采用二进制运算实际上比十进制更简单, 对于加法, 无进位时, $0+0=0$, $0+1=1$, $1+0=1$, $1+1=0$ (带进位), 有进位时, $0+0+$ 进位 $=1$, $0+$

1+进位=0(带进位), 1+0+进位=0(带进位), 1+1+进位=1(带进位), 如:

二进制运算	十六进制运算	十进制运算
(1001 0110) ₂	(96) ₁₆	(150) ₁₀
+ (0101 1010) ₂	+ (5A) ₁₆	+ (90) ₁₀
(1111 0000) ₂	(F0) ₁₆	(240) ₁₀

运算时要记住二进制是逢二进一, 十六进制是逢十六进一。

对于减法, 无借位时, 0-0=0, 1-1=0, 1-0=1, 0-1=1(带借位), 有借位时, 0-0-借位=1(带借位), 1-1-借位=1(带借位), 1-0-借位=0, 0-1-借位=0(带借位), 如:

二进制运算	十六进制运算	十进制运算
(1001 0110) ₂	(96) ₁₆	(150) ₁₀
- (0101 1010) ₂	- (5A) ₁₆	- (90) ₁₀
(0011 1100) ₂	(3C) ₁₆	(60) ₁₀

六、二进制乘除运算

直接使用二进制进行乘除运算同样不难, 读者可以自己验证。

1.1.3 原码、反码与补码

计算机表示一个数, 绝大多数是采用 8 位、16 位或 32 位二进制表示, 一般情况下使用 16 位二进制就可以了, 而如果数的范围很小, 使用 8 位也许就够了, 或者如果要表示数可能很大, 那就采用 32 位, 本书以后若不特别说明, 一般都假定是采用 16 位表示方式。

如果计算机表示的是一个非负整数, 可以很简单地将该数转换为 16 位二进制数即可, 这时, 可以表示的最小数为 0, 最大数为 $2^{16}-1=65\,535$, 二进制转换为十进制只要常规地将每一位转成对应的权值再相加, 这种表示法称为无符号表示法。

可是如果一个数有可能为负数时, 计算机是如何表示这种带符号的数呢? 因为计算机是基于二进制的, 只有 0 和 1, 它是无法直接识别正号“+”与负号“-”的, 计算机使用 0 表示正号“+”, 而使用 1 表示负号“-”, 用于表示正负号的二进制位称为符号位, 一个数的符号位只需一位, 它总是位于其他二进制位的左边, 即符号位是二进制位的最高位。有以下三种表示负数的方案。

一、原码表示法

除符号位以外, 其他二进制位为数值的绝对值, 这种方案称为“原码”表示法, 如:

+20 的原码为: 0 000 0000 0001 0100

↑

0 表示“+”, 其余 15 位二进制值为 20

-20 的原码为: 1 000 0000 0001 0100

↑

1 表示“-”, 其余 15 位二进制值为 -20 的绝对值, 即 20

二、反码表示法

除符号位以外,负数的反码表示是在原码基础上其他二进制位取反,而正数的反码表示与原码相同。如:

```

+20 的反码为: 0 000 0000 0001 0100
                ↑
                0 表示“+”,正数的反码=该数的原码
-20 的反码为: 1 111 1111 1110 1011
                ↑
                1 表示“-”,负数的反码=该数的原码符号位外各位取反

```

也可以认为,负数的反码=对应正数的原码所有位取反。

三、补码表示法

负数的补码表示是在反码基础上加 1,而正数的反码表示与原码相同。如:

```

+20 的补码为: 0 000 0000 0001 0100
                ↑
                0 表示“+”,正数的补码=该数的原码
-20 的补码为: 1 111 1111 1110 1100
                ↑
                1 表示“-”,负数的补码=该数的反码加 1

```

也可以认为,负数的补码=对应正数的原码所有位取反后再加 1。

四、为什么计算机一般采用补码表示法

原码、反码和补码是用于表示负数的三种方案,三种方案中,原码最适合于乘除类运算,补码适合于加减类运算,而反码则加减与乘除都不是最理想,由于加减运算的频率远高于乘除运算,所以多数计算机系统采用的是补码方案,这主要有以下几个原因。

1. 惟一性表示

先计算一下+0 与-0 的表示,如下所示:

```

+0 的编码为: 0000 0000 0000 0000 (原码、反码与补码都相同)
-0 的原码为: 1000 0000 0000 0000
-0 的反码为: 1111 1111 1111 1111
-0 的补码为: 0000 0000 0000 0000

```

-0 的补码等于-0 的反码加 1,这个加 1 为简单加,连符号位同时参与运算,相加时最高位会产生进位,产生的进位位由于超出 16 位而被丢弃,其余的 16 位正好与+0 的补码完全一样,除补码外,其他两种方案中+0 的编码与-0 的编码是不相同的,导致要比较两个数是否相同,不能简单地看两数的 16 位二进制是否完全一样,因此必然要增加硬件上额外开销,这是不必要的。

2. 正负数混合相加

加减法是计算机中最常用的一种运算,它应该尽可能简单,实际上要求加法器在进行

运算时,连同符号位与其他二进制位等同运算,尽量不要特殊化。

我们以 $(-3)+(+1)=(-2)$ 以及 $(-3)+(+5)=(+2)$ 用最简单的二进制加来考察各种编码。

首先看原码方案,应该要有 (-3) 的原码 $+(+1)$ 的原码 $=(-2)$ 的原码。

$$\begin{array}{r} 1000\ 0000\ 0000\ 0011 \\ +\ 0000\ 0000\ 0000\ 0001 \\ \hline 1000\ 0000\ 0000\ 0100 \end{array} \quad \begin{array}{l} (-3)\text{的原码} \\ (+1)\text{的原码} \\ (-4)\text{的原码} \end{array}$$

要求的是加法运算,而实际上通过简单二进制加得到的却是减法的结果,与预期结果不同。

其次看反码方案,应该要有 (-3) 的反码 $+(+1)$ 的反码 $=(-2)$ 的反码。

$$\begin{array}{r} 1111\ 1111\ 1111\ 1100 \\ +\ 0000\ 0000\ 0000\ 0001 \\ \hline 1111\ 1111\ 1111\ 1101 \end{array} \quad \begin{array}{l} (-3)\text{的反码} \\ (+1)\text{的反码} \\ (-2)\text{的反码} \end{array}$$

符合要求,再看另一式子:

$$\begin{array}{r} 1111\ 1111\ 1111\ 1100 \\ +\ 0000\ 0000\ 0000\ 0101 \\ \hline (1)\ 0000\ 0000\ 0000\ 0001 \end{array} \quad \begin{array}{l} (-3)\text{的反码} \\ (+5)\text{的反码} \\ (+1)\text{的反码} \end{array}$$

两数通过简单二进制加时,最高位与其他位一样相加,最高位相加后产生进位,该进位超出 16 位二进制范围,超出部分被舍弃,其余 16 位二进制的结果为+1,与正确结果相差 1,不符合要求。

最后看补码方案,应该要有 (-3) 的补码 $+(+1)$ 的补码 $=(-2)$ 的补码。

$$\begin{array}{r} 1111\ 1111\ 1111\ 1101 \\ +\ 0000\ 0000\ 0000\ 0001 \\ \hline 1111\ 1111\ 1111\ 1110 \end{array} \quad \begin{array}{l} (-3)\text{的补码} \\ (+1)\text{的补码} \\ (-2)\text{的补码} \end{array} \quad (\text{式 } 1)$$

另一式子,应该要有 (-3) 的补码 $+(+5)$ 的补码 $=(+2)$ 的补码。

$$\begin{array}{r} 1111\ 1111\ 1111\ 1101 \\ +\ 0000\ 0000\ 0000\ 0101 \\ \hline (1)\ 0000\ 0000\ 0000\ 0010 \end{array} \quad \begin{array}{l} (-3)\text{的补码} \\ (+5)\text{的补码} \\ (+2)\text{的补码} \end{array}$$

可以看出,只有补码方案才能只使用简单加法器就得到正确答案,其他方案的加法器都需要一定的变换或补偿。

3. 带符号与无符号的混合相加

除带符号的数以外,计算机中还有大量的数是使用无符号编码方案,带符号数相加、无符号数相加以及带符号数与无符号数的相加,如果采用补码方案,一个简单加法器能满