面向系统能力培养大学计算机类专业规划教材

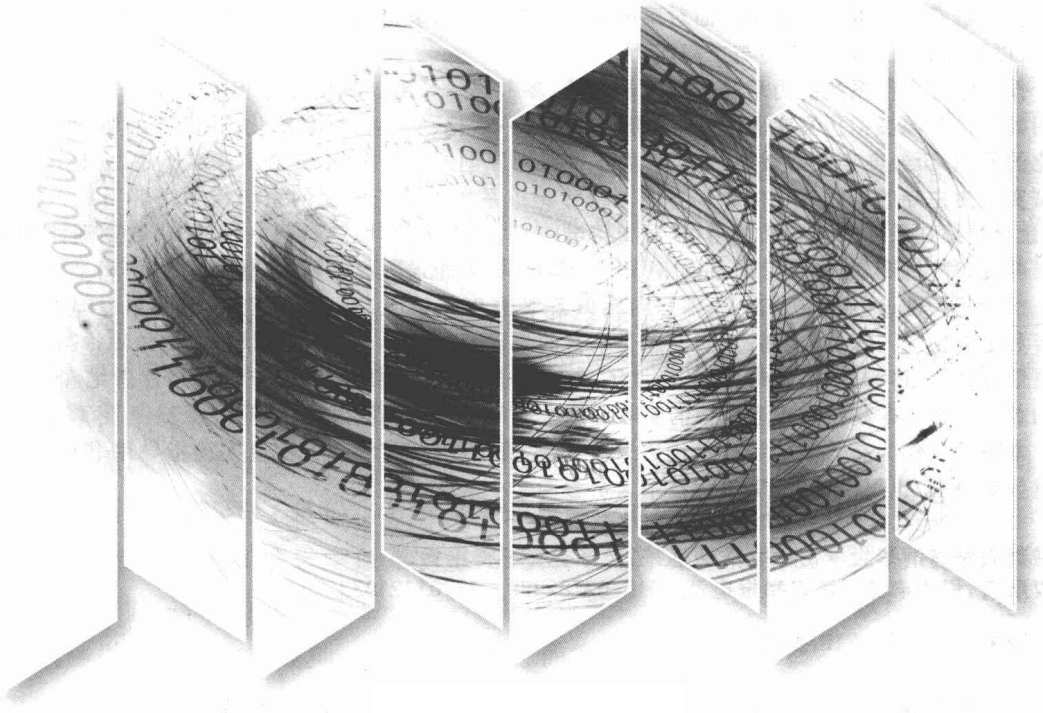
计算机组成原理实践教程

——从逻辑门到CPU

谭志虎　秦磊华　胡迪青　编著

清华大学出版社

面向系统能力培养大学计算机类专业规划教材



计算机组成原理实践教程

——从逻辑门到CPU

谭志虎　秦磊华　胡迪青　编著

清华大学出版社
北京

内 容 简 介

“计算机组成原理”是一门理论性、实践性、工程性和技术性都很强的专业核心基础课程，其理论教学与实践教学具有同等重要的地位。面对系统能力培养的新要求，有必要对计算机组成原理的实践内容进行再设计，以强化系统级设计能力与工程能力。

作者结合多年系统能力培养实践教学经验，参考国际一流计算机专业相关课程的先进经验，引入易学易用的免费开源虚拟仿真实验平台，本着理论实践一体化、实验目标系统化、课内课外协同化、教师指导轻松化的原则，历经5年实践教学改革，开发了一系列原创的课程实验，建立了逐层递进、立足计算机系统、设计型实验为主导的实践教学体系。本书围绕计算机系统设计与实现的系统能力培养目标，通过一系列实验，让学生站在硬件工程师的视角，从逻辑门电路开始，逐步设计运算部件、存储器、数据通路和控制器、流水线冲突冒险机制，直至设计完整的MIPS流水CPU来深入理解计算机软硬件系统。

实验紧扣理论教学重点难点，课程内容覆盖率达90%以上，有助于构建理论课堂中师生的共同语境。内容和难度由浅入深、层次递进，实验配备丰富的自动评测工具，为学生提供精准的故障诊断。本课程的及时反馈好，学生体验感好，获得感强，激发了学生参与实践的积极性，有助于学生硬件思维和计算机系统能力的养成。在985高校和普通院校的实际应用中取得了很好的效果。

本书适合作为高等学校计算机相关专业“计算机组成原理”课程的配套实验指导用书和系统能力培养的重要参考书，也可供相关IT工程技术人员参考。

图书在版编目(CIP)数据

计算机组成原理实践教程：从逻辑门到CPU/谭志虎，秦磊华，胡迪青编著. —北京：清华大学出版社，2018
（面向系统能力培养大学计算机类专业规划教材）
ISBN 978-7-302-51189-2

Ⅰ. ①计… Ⅱ. ①谭… ②秦… ③胡… Ⅲ. ①计算机组成原理－高等学校－教材 Ⅳ. ①TP301

中国版本图书馆CIP数据核字(2018)第209921号

责任编辑：张瑞庆
封面设计：常雪影
责任校对：焦丽丽
责任印制：杨　艳

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn，010-62795954
印 装 者：大厂回族自治县正兴印务有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：17　　**字　　数**：413千字
版　　次：2018年10月第1版　　**印　　次**：2018年10月第1次印刷
定　　价：45.00元

产品编号：069480-01

FOREWORD

前　言

“计算机组成原理”作为计算机专业核心基础课，主要讨论计算机各大功能部件的基本组成原理及其互连构成整机的技术，在计算机系统能力培养目标中起着重要的承上启下作用，其先导课程包括“数字逻辑”和“汇编语言程序设计”，后续课程为“计算机系统结构”和“计算机接口技术”。该课程的实践教学应该站在贯穿计算机硬件系列课程的角度，引导学生如何利用前导课程的基本知识，设计计算机主要功能部件并构成整机系统。与理论课程配合，实现计算机系统设计能力的培养 。

“计算机组成原理”课程理论综合性强，课程实验尤其是综合实验难度大，存在时间、场所、平台等诸多制约因素，为了从根本上解决实践教学中的困难，作者结合多年的教学经验，进行了一系列卓有成效的实践教学改革。参考国际一流计算机专业相关课程的先进经验，引入了易学易用的开源虚拟仿真实验平台，本着理论实践一体化、实验目标系统化、实验平台虚拟化、教师指导轻松化的原则，历经 5 年持续的实践教学改革，开发并完善了一系列内容丰富的课程实验，建立了逐层递进、立足计算机系统、设计型实验为主导的实践教学体系。本教材就是相关实践教学改革的主要成果，其主要特点如下。

(1) 理论实践一体化：相关实验紧扣理论教学重点、难点，课程内容覆盖率达 90％以上，有助于构建理论课堂中师生的共同语境，为翻转课堂提供丰富的教学素材，实验内容大多处于布鲁姆认知分类法中的运用、分析、综合、评价 4 个层次，学生完成系列实验，对相关知识的掌握更透彻，为理论教学提供强力的支撑。

(2) 实验目标系统化：围绕计算机系统设计与实现的系统能力培养目标，让学生站在硬件工程师的视角，从逻辑门电路开始，逐步设计运算部件、存储器、数据通路和控制器、冲突冒险机制，直至设计完整的流水 CPU 来深入理解计算机软硬件系统，将系统能力培养的复杂工程问题——五段流水 CPU 设计的高难度任务分解成若干难度递进的子任务和单元实验。

(3) 实验平台虚拟化：主要实验均采用跨平台开源的 Logisim 虚拟仿真平台，该平台简单易学、易调试且无实验成本，突破传统实验对空间和时间的要求。美国加州大学伯克利分校 CS61C、康奈尔大学 CS3410 课程均采用了该平台。实验平台采用分离数字电路对象构建原理图的方式进行 CPU 系统设计和仿真，有效地延续了数字逻辑课程的设计方法，有利于培养学生硬件设计思维。无须另外开设硬件描述语言类先导课程，也回避了硬件描述语言过于抽象、硬件设计程序化、学习周期长的问题。

(4) 实验过程游戏化：借鉴游戏闯关的设计理念，将具有高难度挑战性的实验总目标细分成若干可明确检查的学习关卡。实验内容和难度逐渐递进，通过各阶段成果实时得分增强学习的及时反馈，提高学习趣味性；为学生提供丰富的测试用例以及自动检测工具等学习

FOREWORD

装备，让学生在较短时间内体验更多的设计内容；实验完成率高，学生成就感强，有助于激发学生的持续学习热情，在985高校和普通院校的实际应用中均取得了很好的效果。

(5) 教师指导轻松化：传统硬件系列实验检查和指导难度大，教师劳动强度高，通过实验输入输出界面规范化、测试用例标准化、性能指标数据化、实验测试评分自动化等一系列手段，提升了实验的易检查性，大大降低了教师工作负担。

本书前7章按“计算机组成原理”各章节内容进行组织，每章均结合各章的重点、难点设置若干实验，各实验均明确给出学生的学习目标和实验内容，并进行适当的背景知识补充，实验思考部分可有效地引导学生进行实验。其中，第1章为数据表示实验，设计了若干程序运行验证性实验，有助于学生站在软硬协同的角度理解数据表示；另外还设计了汉字编码、海明编校验码、CRC校验码实验，其中流水传输实验创造性地模拟了一个编码传输环境，让学生理解编码传输概念的同时，提前熟悉流水线的相关知识。第2章为运算器组成实验，主要包括先行进位加法器实验、MIPS运算器设计实验、阵列乘法器设计实验、原码及补码1位乘法器设计实验。第3章为存储系统实验，主要包括存储扩展实验、MIPS RAM设计实验、MIPS寄存器文件设计实验、虚拟存储器软件仿真实验，另外还包括3个Cache相关的实验，既包括Cache性能仿真，也包括Cache硬件设计实验，填补了组成原理实验在这方面的空白。第4章为MIPS汇编程序设计实验，主要介绍MIPS指令系统以及MIPS汇编程序设计方法，并给出若干MIPS汇编程序设计实验。第5章MIPS处理器设计实验为课程实验重点，包括MIPS单周期处理器设计实验、MIPS多周期处理器硬布线控制器设计实验、多周期处理器微程序控制器设计实验、基于FPGA的单周期处理器实验。第6章为MIPS五段流水机制设计实验，包括理想流水线CPU设计实验、气泡流水线CPU设计实验、重定向流水线CPU设计实验、动态分支预测机制设计实验等。第7章为输入输出系统实验，包括中断与轮询方式编程实验、单级中断机制设计实验、多级中断机制设计实验、流水中断机制设计实验等。第8章介绍虚拟仿真平台Logisim使用说明，方便师生快速掌握Logisim使用技巧。第9章给出Logisim中常用组件库参考手册，介绍不同组件的详细使用信息。第10章简要介绍MIPS汇编仿真器MARS的主要功能和使用方法。

注意：本书图中的逻辑电路符号与平台软件中的符号一致。

本书实验众多，内容丰富，教学过程中教师可以根据学生具体情况有针对性地选择部分内容开展实验，学生也可以根据自己的学习情况自行选择实验。本书适合作为高等学校计算机相关专业“计算机组成原理”课程的实验教材，也可供IT工程技术人员参考。

特别感谢华中科技大学计算机科学与技术学院秦磊华教授，本书正是在秦教授的策划和不断鞭策鼓励下才得以完成；感谢华中科技大学计算机卓越1201班的全体学生，是他们

和老师的高度配合才有了今天的一系列成果，尤其感谢王汉杰、徐钦振对 Logisim 手册的翻译工作；感谢华中科技大学计算机学院 2013 级、2014 级、2015 级全体学生，本书大多数实验均经过了他们多次检验和持续改进；最后感谢在我身后默默支持的家人，谢谢他们！

由于水平有限，书中难免存在错误和疏漏之处，敬请同行和广大读者批评指正，邮箱：stan@hust.edu.cn。

作　者

2018 年 7 月

CONTENTS

目 录

CONTENTS

CONTENTS

第1章　数据表示实验

1.1　补码表示实验

1.1.1　实验目的

理解补码的基本特性，能分析简单C语言程序中与数据表示相关的执行行为，并通过实际代码执行验证真实计算机中数据表示的编码体系，掌握获取C语言变量机器码的方法，培养数据表示层面基本的软硬件协同系统观。

1.1.2　背景知识

人类历史上第一台数字计算机ENIAC采用十进制进行运算，目前计算机普遍采用二进制进行数据表示和运算，二进制可以表示任何数据和信息，而且状态数最少，运算简单，便于物理实现。二进制数与十进制数一样有正负之分，日常生活中采用“+”或“−”表示的数据称为真值，在计算机中，通常将数的符号和数值一起编码来表示数据，这种用“0”或“1”表示数据符号位的编码称为机器码，常用的机器码分为原码、反码、补码和移码。

设 x 为 n 位定点数(不含符号位)，则 x 的原码、反码、补码和移码的定义如表1.1所示。

表1.1　4种机器码定义

机器码	定点整数 当$-2^n<x\leqslant 0$时	定点小数 当$-1<x\leqslant 0$时	当$0\leqslant x\leqslant 2^n-1$， $0\leqslant x\leqslant 1-2^{-n}$时	备　注
原码	$2^n+\lvert x\rvert$	$2+\lvert x\rvert$	x	最直观，增加一个符号位
反码	$2^{n+1}+x-1$	$2+x-2^{-n}$	x	符号位同原码，为负数时数值位逐位取反
补码	$2^{n+1}+x$	$2+x$	x	反码末位加1得到补码，多表示一个负数
移码	2^n+x	无	2^n+x(整数时)	补码符号位取反得到移码

(1) 原码表示最为直观，仅仅是在数据位前面增加一个符号位，“0”表示正数，“1”表示负数，其在数轴上表示区间对称，存在正零和负零两个零，运算比较复杂，目前在计算机中主要用于表示浮点数的尾码部分，所以浮点数存在正零和负零两个不同的编码。

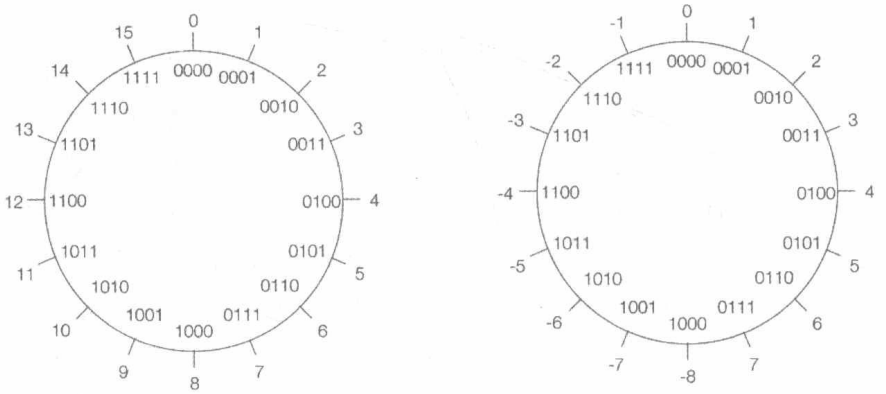
(2) 反码符号位与原码相同，数值表示范围也相同。当真值为正数时，数值部分与真值相同；当真值为负数时，数值部分可以通过对真值逐位取反得到，反码末位加1即可得到补码。由于反码在二进制电路中非常容易得到，所以反码通常用于补码减法运算中减数的求补过程。

(3) 补码数据表示建立在“模”的概念基础上，模的值即为符号位进位所在位的权值。

对于定点小数而言，模的值为 2^1（即 2），对于定点整数而言，模的值为 2^{n+1}，其中 n 为不包含符号位的整数位数。当真值为负数时，补码等于真值加上模，补码这种特性使得补码符号位可以直接参与加减运算，运算电路实现方便，另外补码只有唯一的零，所以在表示范围上比其他几种机器码多表示一个最小负数。目前，计算机中整数采用补码进行存储、表示和运算。

(4) 移码主要用于表示浮点数的阶码，IEEE754 浮点数标准中使用的移码和表 1.1 中略有差异。

在 C 语言中，整型数据分为无符号和有符号两种。为便于描述，图 1.1 给出了 4 位整数表示的循环圈，实际上 32 位整型变量原理也完全相同。图 1.1(a)为无符号数循环圈，从图中可以看出，真值和二进制机器码相等，当两个数相加结果超过 $2^4-1=15$ 时，结果将在循环圈顶部沿顺时针方向循环，运算结果将小于相加数，产生溢出。同理当两个数据相减结果小于零时，结果将在循环圈顶部沿逆时针方向循环，运算结果大于被减数，产生溢出。



(a) 4位无符号数　　(b) 4位有符号数(补码)

图 1.1　4 位整数表示循环圈

图 1.1(b)是采用补码表示的 4 位有符号数表示循环圈，当真值为正数时，二进制的机器码和十进制的真值相同，−1 的机器码是全 1，补码公式为 $2^4-1=1111$（模为 2^4），−2 的机器码为 $2^4-2=1110$。最大正数是 $2^4-1=0111$，两个正数相加结果如果超过这个数值，则会在循环圈底部顺时针进入负数区域，运算结果不正确，产生正溢出。最小负数是 $-2^3=-8$，两个负数相加结果如果小于最小负数，则会在循环圈顶部顺时针进入正数区域，结果会变成正数，运算结果不正确，产生负溢出。实际上，C 语言程序并不检测加、减、乘等运算的溢出，程序员应尽量避免出现这种情况，对溢出应通过程序进行判断。

8 位、16 位、32 位、64 位的整型变量基本原理与上面的 4 位整数原理完全一样，其对应取值范围见表 1.2。

表 1.2　整型变量取值范围

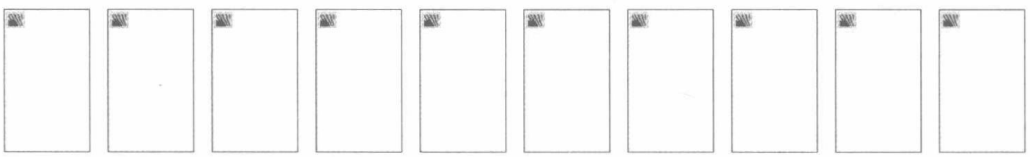
Value		char(8 位)	short(16 位)	int(32 位)	long(64 位)
无符号最大值	Hex	0xFF	0xFFFF	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF
	Dec	255	65 535	4 294 967 295	18 446 744 073 709 551 615

续表

Value		char(8 位)	short(16 位)	int(32 位)	long(64 位)
有符号最小值	Hex	0x80	0x8000	0x80000000	0x8000000000000000
	Dec	−128	−32 768	−2 147 483 648	−9 223 372 036 854 775 808
有符号最大值	Hex	0x7F	0x7FFF	0x7FFFFFFF	0x7FFFFFFFFFFFFFFF
	Dec	127	32 767	2 147 483 647	9 223 372 036 854 775 807
−1	Hex	0xFF	0xFFFF	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF
0	Hex	0x00	0x0000	0x00000000	0x0000000000000000

1.1.3　实验内容

(1) 小练习：如果给你 10 个信封和 1000 张一元的纸币，如何将 1000 元钱分装到 10 个信封中才能保证不管给出 1～1000 的任何一个数字，都可以从 10 个信封中直接拿走若干信封，而拿走信封里面的钱的总数正好等于这个数字？请在下面 10 个信封上写上具体钱数。



(2) 仔细阅读如下 C 语言代码，结合数据表示相关知识，分析程序功能。

```
//file: machine_code_output.c
#include "stdio.h"
//输出字符的十六进制编码
void char_hex_out(char a)
{
  const char HEX[]="0123456789ABCDEF";
  int index=a&0x0F;
  printf("%c%c", HEX[(a&0xF0)>>4],HEX[a&0x0F]);
}
//输出 4 个字节数据的十六进制编码，可用于输出 4 字节变量的机器码
void four_byte_out (char *addr)
{
  char_hex_out (*(addr+3));    //输出指针变量的值，指针本质上是内存地址，无符号数
  char_hex_out (*(addr+2));   // 假设采用小端方式存储
  char_hex_out (*(addr+1));
  char_hex_out (*(addr+0));
  printf("\n");
}
main()
{
```

```
  int a=-1;
  int b=2147483648;          //2147483648=2^31    4294967296=2^32
  int c=-b;
  unsigned int d=-2147483648;
  printf ("a=%u=%d=0x%x  \n",a,a,a);
  printf ("b=%u=%d=0x%x  \n",b,b,b);
  printf ("c=%u=%d=0x%x  \n",c,c,c);
  printf ("d=%u=%d=0x%x  \n",d,d,d);

  printf("\nd's memory addr=0x%x",&ui);     //输出变量 d 的虚存地址
  printf("\nd's machine code=0x");
  four_byte_out (&ui);     //输出变量 d 的机器码

  return;
}
```

上述代码输出如下，请根据自己的分析和理解完成程序的输出。

a=_____=_____=0x_____

b=_____=_____=0x_____

c=_____=_____=0x_____

d=_____=_____=0x_____

d's memory addr=0x_____

d's machine code=0x_____

在C语言程序开发环境中编译运行以上代码，验证你自己的分析结果，如有不同，请思考原因，并找出依据。

(3) 仔细阅读如下C语言代码，结合数据表示相关知识，分析程序功能。

```
//file: int_add_overflow.c
#include "stdio.h"
main()
{
  int a=1; int b=2147483647;          //2147483648=2^31    4294967296=2^32
  int d=-1; int e=-2147483648;
  int c=a+b;
  int f=e+d;
  printf ("c=a+b=%d+%d=%d  \n",a,b,c);
  printf ("f=e+d=%d+%d=%d  \n",d,e,f);
  return;
}
```

上述代码输出如下，请根据自己的分析和理解完成程序的输出。

c=a+b=_____+_____=_____

f=e+d=_____+_____=_____

在C语言程序开发环境中编译运行以上代码，验证你自己的分析结果，如有不同，请思

考原因，并找出依据。

(4) 仔细阅读如下C语言代码，结合数据表示相关知识，分析程序功能。

```
//file: get_var_size.c
#include "stdio.h"
#include "stdlib.h"
int main ()
{
  int iarray[]={5, 4, 3, 2, 1};
  char * iarr=( char *) iarray ;
  printf (" sizeof ( char ): %d\n", ( int) sizeof ( char ) ) ;
  printf (" sizeof ( short ): %d\n", ( int ) sizeof ( short ) ) ;
  printf (" sizeof (int ): %d\n", ( int) sizeof (int ) ) ;
  printf (" sizeof ( unsigned int ): %d\n", (int ) sizeof ( unsigned int ) ) ;
  printf (" sizeof ( long ): %d\n", ( int) sizeof ( long ) ) ;
  printf (" sizeof ( long long ): %d\n", (int ) sizeof ( long long ) ) ;
  printf (" sizeof ( size_t ): %d\n", (int ) sizeof ( size_t ) ) ;
  printf (" sizeof ( void * ): %d\n", (int ) sizeof ( void * ) ) ;
  printf (" sizeof ( iarray ): %d\n", (int ) sizeof ( iarray ) ) ;
  printf (" sizeof ( iarr ): %d\n", ( int) sizeof ( iarr ) ) ;
  return 0;
}
```

上述代码输出如下，请根据自己的分析和理解完成程序的输出。

```
sizeof(char):_____
sizeof(short):_____
sizeof (int):_____
sizeof(unsigned int):_____
sizeof(long):_____
sizeof(long long):_____
sizeof(size_t):_____
sizeof(void * ):_____
sizeof(iarray):_____
sizeof(iarr):_____
```

在C语言程序开发环境中编译运行以上代码，验证你自己的分析结果，如有不同，请思考原因，并找出依据。

1.1.4 实验思考

(1) 在实验内容(2)的程序中，变量b和c是否相等，为什么？

(2) 在实验内容(2)的程序中，变量d的内存地址输出的值是否为一个确定的值，其值能否被4整除，为什么？

(3) C语言中int、short、char、long型变量在内存中存储的机内码是采用什么编码？

(4) C语言中int变量是采用小端方式存储还是大端方式存储？小端方式存储和大端

存储方式由什么来决定?

1.2 浮点数表示实验

1.2.1 实验目的

理解 IEEE754 编码基本规则和特点,能分析程序中浮点运算的执行结果,并利用程序实际执行验证真实计算机中浮点数据的表示规则,了解 IEEE754 编码表示容易引起的一些问题。

1.2.2 背景知识

IEEE754 浮点数标准是目前主流计算机所采用的浮点数标准,主要包括 32 位单精度浮点数和 64 位双精度浮点数,分别对应 C 语言中的 float 型和 double 型。IEEE754 浮点数据由符号位 S、阶码 E 和尾数 M 组成,具体形式如图 1.2 所示。

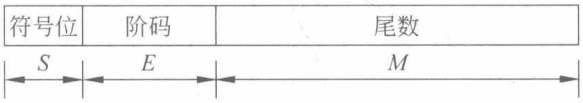


图 1.2 IEEE754 浮点数格式

单精度浮点数共 32 位,其中 S 占 1 位,E 占 8 位,M 占 23 位;双精度浮点数共 64 位,其中 S 占 1 位,E 占 11 位,M 占 52 位。

阶码字段 E,单精度浮点数采用偏移值为 127 的移码表示,双精度浮点数阶码偏移值为 1023。尾数字段 M 小数点左边包含一位隐藏位 1,即尾数的实际有效位数为 24/53 位,完整的尾数形式为 1.M,但在进行浮点数据表示时只保存 M。正是由于要将位数变成 1.M,阶码的偏移值才为 127/1023,而不是 128/1024。

随着 E 和 M 的取值不同,IEEE754 浮点数据表示具有不同的意义,表 1.3 给出了 IEEE754 单精度浮点数在不同 S、E、M 取值下的具体表示意义。

表 1.3 IEEE754 单精度浮点数规范

符号位 S(1 位)	阶码 E(8 位)	尾数 M(23 位)	表　示
0/1	255	1××××	NaN 非数
0/1	255	0××××	sNaN 发信号的非数(尾数不全为零)
0	255	0	$+\infty$
1	255	0	$-\infty$
0/1	1～254	M	$(-1)^S \times 1.M \times 2^{e-127}$ 规格化数
0/1	0	M(非 0)	$(-1)^S \times 0.M \times 2^{-126}$ 非规格化数
0/1	0	0	+0/−0

从表 1.3 可以看出,浮点数可以表示正无穷和负无穷数,另外还可以表示非数 NaN。为

进一步提升浮点数表示精度，还引入了非规格化数，浮点数尾码采用原码表示，所以存在正零和负零两个零，单精度浮点数、双精度浮点数表示范围见表 1.4。

表 1.4　IEEE754 单精度浮点数、双精度浮点数表示范围

格　式	最 小 值	最 大 值
单精度规格化 $(-1)^S \times 1.M \times 2^{e-127}$	$E_{min}=1, M=0$, $1.0 \times 2^{1-127}=2^{-126}$	$E_{max}=254, M=1111\cdots1$ $f=1.1111\cdots1 \times 2^{254-127}=2^{127} \times (2-2^{-23})$ $\approx +3.4 \times 10^{38}$
单精度非规格化 $(-1)^S \times 0.M \times 2^{-126}$	$E=0$, $M=2^{-23}$, $2^{-23} \times 2^{-126}=2^{-149}$	$E=0, M=1111\cdots1$ $f=0.1111\cdots1 \times 2^{-126}=2^{-126} \times (1-2^{-23})$
双精度规格化 $(-1)^S \times 1.M \times 2^{e-1023}$	$E_{min}=1$, $M=0$, $1.0 \times 2^{1-1023}=2^{-1022}$	$E_{max}=2046, M=1111\cdots1$ $f=1.1111\cdots1 \times 2^{2046-1023}=2^{1023} \times (2-2^{-52})$ $\approx +1.8 \times 10^{308}$
双精度非规格化 $(-1)^S \times 0.M \times 2^{-1022}$	$E=0$, $M=2^{-52}$, $2^{-52} \times 2^{-1022}=2^{-1079}$	$E=0, M=1111\cdots1$ $f=0.11111\cdots1 \times 2^{-1022}=2^{-1022} \times (1-2^{-52})$

IEEE754 单精度浮点数与对应真值之间的变换流程如图 1.3 所示。

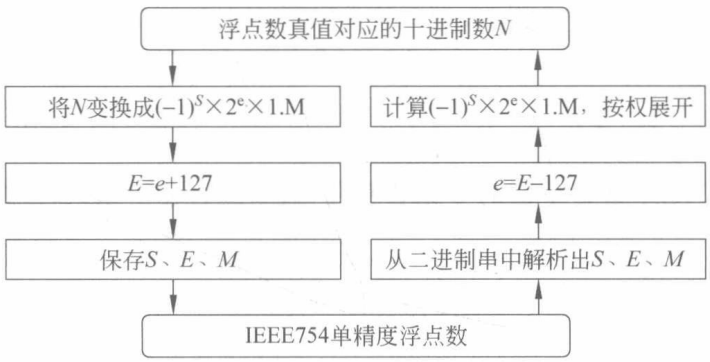


图 1.3　IEEE754 单精度浮点数与真值间的变换流程

值得注意的是，十进制小数大多不能精确转换成二进制数，如 0.1、0.2、0.3、0.4 等在转换成二进制小数时都会变成循环小数，所以即使有再多的尾数位也无法精确表示这些十进制数，浮点数并不能精确表示所有十进制数。另外，由于 IEEE754 浮点数采用阶码和尾码的形式表示浮点数，所以浮点数在数轴上的刻度分布并不像定点整数一样是均匀的，图 1.4 是单精度浮点数在数轴上的刻度分布，零右边的阴影区是非规格化数区域，越往右浮点数的分布就越稀疏，这也导致浮点数的加法运算不满足结合律，小数和大数相加时小数可能会被吸收。

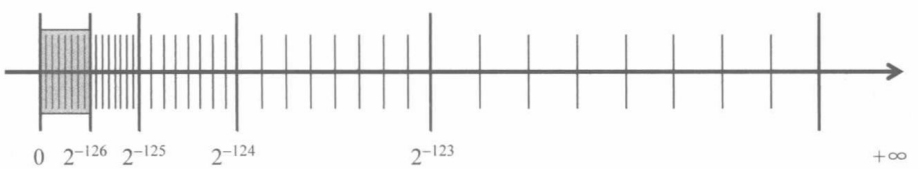


图 1.4　IEEE754 单精度浮点数在数轴上的刻度分布

1.2.3 实验内容

(1) 小练习：假设你已经身家千万，请问：你最少有多少钱(以元为单位的整数)才能导致32位的单精度浮点数无法精确表示？你最少有多少钱才能导致64位的双精度浮点数无法精确表示，写一个小程序将这两个数字用十进制打印出来，同时请精确输出32位单精度浮点数的最小非规格化数、最大非规格化数的十进制值。

(2) 无法表示的整数：运行下面的C语言程序，观察运行输出结果，并解释运行现象。

```
//file: unrepresentable .c
#include<stdio .h>
#include<stdlib .h>
int main(int argc, char * * argv)
{
    float f1=16777216.0;
    float f2=16777217.0;
    float f3=16777218.0;
    printf (" 16,777,216: %f\n",f1);
    printf (" 16,777,217: %f\n",f2);
    printf (" 16,777,218: %f\n",f3);
    printf ("f1==f2? %s",f1==f2 ? " true " : " false ");
    return 0;
}
```

(3) 整数除零问题：运行下面的C语言程序，观察运行输出结果，并解释运行现象。

```
//file: int_div_by_zero.c
#include "stdio.h"
main()
{
  int a=1; a=a/0;
  printf("a=%d",a);
  return;
}
```

(4) 浮点数除零问题：运行下面的C语言程序，观察运行输出结果，并解释运行现象。

```
//file: float_div_by_zero.c
#include "stdio.h"
main()
{
  float a=0.0, b;
  a=a/0;  b=-sqrt(-1);
  printf ("a=%f b=%f",a,b);
  return;
}
```