电子科技大学本科规划教材

微计算机原理与接口

——基于STM32处理器

WEIJISUANJI
YUANLI YU JIEKOU

陈客松 汪 玲 庞晓凤 编著

电子科技大学出版社

电子科技大学本科规划教材

微计算机原理与接口

——基于 STM32 处理器

陈客松 汪 玲 庞晓凤 编著

电子科技大学出版社

图书在版编目（CIP）数据

微计算机原理与接口：基于STM32处理器／陈客松，汪玲，庞晓凤编著. —成都：电子科技大学出版社，2017. 7

ISBN 978-7-5647-4766-4

Ⅰ. ①微… Ⅱ. ①陈… ②汪… ③庞… Ⅲ. ①微型计算机-理论-高等学校-教材②微型计算机-接口技术-高等学校-教材 Ⅳ. ①TP36

中国版本图书馆CIP数据核字（2017）第154028号

微计算机原理与接口——基于STM32处理器

陈客松　汪　玲　庞晓凤　编著

策划编辑　刘　凡

责任编辑　刘　凡

出版发行　电子科技大学出版社

成都市一环路东一段159号电子信息产业大厦九楼　邮编：610051

主　　页　www.uestcp.com.cn

服务电话　028-83203399

邮购电话　028-83201495

印　　刷　成都市火炬印务有限公司

成品尺寸　185mm×260mm

印　　张　24.75

字　　数　640　千字

版　　次　2017年7月第1版

印　　次　2017年7月第1次印刷

书　　号　ISBN 978-7-5647-4766-4

定　　价　60.00元

前言
PREFACE

近三十年来，国内高等院校的工程类专业，诸如电子信息工程、集成电路、自动化、机械电子工程等非计算机专业的“微型计算机系统原理及接口技术”或“微机原理与接口技术”课程，一直沿用 Intel 8086/8088 CPU 及其外围芯片作为主要教学内容，国内外出现过不少 x86 架构的计算机原理和接口的经典教材，其知识体系和教学内容成熟而且适用。然而随着计算机技术的突飞猛进，这些教材逐渐在近十年来的教学过程中受到业界越来越多的质疑和诟病。主要原因是知识点已经过于陈旧和不足，基础知识和技术概念亟待更新。

当前有大量的微计算机新技术、新器件、新应用等待学生去学习和掌握。当然，业界看重的基本原理和基础知识确实也很重要，不过这些基础性的内容并非 8086CPU 所独有的。当代 CPU 中所包含的基本工作原理和基础知识已经比 8086 更加全面和更加深刻，是到了引入更先进的处理器来介绍微计算机基本知识和工作原理的时代了。

本书以 ST 公司的 STM32 处理器为蓝本介绍微处理器体系结构、指令系统及汇编/C 语言程序设计、接口应用技术等内容。在大幅度更新课程内容的同时，保持了课程结构的稳定。ST 公司的 STM32 处理器在技术上是先进、成熟和实用的，能够学以致用；它有比较宽广的应用范围，能够承载学生大量的理论学习和实践活动；有较低的应用成本和学习门槛，有利于实践训练和快速入门，期望能起到抛砖引玉的作用。

全书分 11 章，第 1 章主要介绍微型计算机系统的基础知识；第 2 章介绍计算机的基本结构和工作原理；第 3 章介绍 ARM 微处理器；第 4 章介绍 STM32 微处理器；第 5 章介绍 Cortex-M3 指令系统；第 6 章介绍汇编程序设计；第 7 章介绍存储器及其接口技术；第 8 章介绍输入/输出接口；第 9 章介绍总线；第 10 章介绍中断系统；第 11 章介绍 ARM 嵌入式计算机系统设计，并给出了一个应用实例。

本书既重视微型计算机的基本概念和基本原理，又重视接口技术的实际应用，二者协调一致，相得益彰。可作为高等院校电子信息工程、通信工程、电子科学与技术、集成电

路、自动化等专业的教材，也可作为 ARM 技术培训教材和有关人员的自学教材。建议课内 72 学时，其中授课 56 学时，上机实践 16 学时。

参加本书编写的教师多年来一直从事“计算机系统原理”“微型计算机系统原理与接口技术”和“软件技术基础”等课程的理论与实践教学，同时也有一定的项目开发经验。本书由陈客松组织编写和统稿，完成了第 1-3、第 11 章的编写，第 4 章由陈客松和庞晓凤共同编写、第 5、6 章由庞晓凤编写，第 7-10 章由汪玲编写。

本书的出版得到了电子科技大学教务处、出版社的大力支持。编著本书时参考了国内外大量著作、文献和网络资源，难以一一列出。同课程组的任课老师提出了宝贵的建设性意见，研究生郭瑞、巩云龙、石佳佳、李亚飞和姜金男完成了部分编辑工作，在此一并表示衷心感激！鉴于作者水平有限，书中难免存在不足和失误之处，恳请和希望读者提出宝贵意见和建议，以便再版时改进。

编著者

2017 年 5 月

目 录
CONTENTS

第 1 章　微型计算机系统的基础知识

计算机又称电脑，是一种用于高速计算的电子机器，可以进行数值计算、字符处理和逻辑运算，还具有存储记忆功能。计算机是能够按照程序运行，自动、高速处理海量数据的现代化智能电子设备。计算机按照性能或应用领域可分为超级计算机、工业控制计算机、网络计算机、个人计算机、嵌入式计算机五类，较先进的计算机有生物计算机、光子计算机、量子计算机等。

计算机是20世纪人类最先进的科学技术发明之一，对人类的生产活动和社会活动产生了极其重要的影响，并以强大的生命力飞速发展。它的应用领域从最初的军事科研应用扩展到社会的各个领域，现已形成了规模巨大的计算机产业，带动了全球范围的技术进步，由此引发了深刻的社会变革。如今计算机已遍及一般学校、企事业单位，进入寻常百姓家，成为信息社会中必不可少的工具。

计算机是非常“确定”的一个同步时序逻辑系统，所谓“确定”，即在任何时候，在相同的方法、相同的状态下(当然还包括相同的起始条件)，同样的问题必然获得相同的结果。计算机本身是没有心智的，不是什么具有思维能力的神秘、高级生物，它只是一个机器，只会精确地按照人的要求去执行任务。在计算机按照性能指标的分类中，除了巨型机、大型机、中型机和小型机外都是微型计算机，包括台式机、笔记本电脑以及各种嵌入式计算机等，简称为微计算机。

本章主要介绍计算机进行运算的相关基础知识，概述微型计算机系统的组成，介绍微型计算机的硬件系统、软件系统和计算机的发展简史。

1.1　计算机的运算基础

自然界和社会生活中广泛存在着两类显著对立的事物，譬如黑与白、火与水、真与伪，正如电子计算机内有低电平和高电平两个状态对立的物理量，分别被人们抽象地标记为信息“0”和“1”。计算机就是记忆、存储、转换、处理和传输这类信息的系统。

计算机处理的对象可以统称为数据，其中具有数值大小和正负特性的数据称为数值数据，其他表示文字、图形、声像一类的信息并无大小和正负特性的数据统称为非数值数据，它们在计算机内部是以不同形式编码的“0”和“1”的二进制数(数值数据)或“0”和“1”组合编码的符号串(非数值数据)来表示和存储的。计算机内部所有的数字逻辑部件都只能“识别”由0、1组成的二进制形式的数据，因此人们在现实世界中所使用的任何形式的信息，不论是数字、文字、声音、图像，还是动画与视频等其他类型的信息，它们都必须转换成二进制数形式表示以后，才能在计算机内部进行存储、计算、处理和传输。

这里主要介绍各种数制之间相互转换的综合表示法、常见的二进制编码以及有关补码溢出等几个问题。

1.1.1 计算机中数值信息的表示与运算

本节介绍计算机中用于表示数值信息的常见术语，包括二进制编码、机器数与真值、原码、反码、补码、定点数与浮点数。

1. 二进制数据及其编码

由于计算机只能识别二进制数，所以，输入的信息，如数字、字母、符号等都需要用特定的二进制码来表示，这就是二进制(Binary)编码。二进制编码只用“0”和“1”两个符号。位 b(bit)是计算机系统所能表示的最基本、最小的数据单位，也就是二进制数中的一个数值位。计算机系统处理的数据对象可分为多种数据类型，如位 b(bit)、字节 B(Byte,8bits 为 1Byte)、字 W(Word)、双字 DW(Double Word)、4 字 QW(Quad Word)等。一个位只有两种取值情况，即“0”或“1”，对应的是硬件电路中单根导线上“低”电平或“高”电平的状态。

数据用二进制位表达时，仍然按日常书写习惯：低位在右边、高位在左边。20 世纪 80 年代，IBM 系列机以 16 位结构的 Intel 8086 和 80286 作为处理器并获得广泛应用，所以 Intel 80x86 系列处理器以 16 位二进制数据为一个字，这样，32 位称为双字，64 位称为四字。

近年来，随着计算机理论和技术的快速发展，“字”作为一种数据类型，其含义已经发生了变化。新的更高位数的微处理器不断涌现，为了表述和使用方便，不再以 16 位为一个字，而是定义 32 位二进制数据为一个字，譬如在 ARM 体系结构中，字类型的数据是 4 个字节，把 64 位二进制数据称为双字类型的数。

二进制表示的数的最低位常称为最低有效位(Least Significant Bit,LSB)，即 D0 位；最高位则称为最高有效位(Most Significant Bit,MSB)。对应 x86 系列处理器的字节、字、双字和四字数据，MSB 依次是 D7、D15、D31 和 D63。二进制表达不直观且不方便，所以通常使用十六进制(Hexadecimal)，它容易与二进制相互转换。汇编语言常使用后缀字母 H(大小写均可)表示十六进制数据，高级语言通常用前缀 0x 表示十六进制数据。二进制数据用后缀字母 B(大小写均可)或前缀 0b 表示。一个十六进制位对应 4 个二进制位，即 0H=0000B，lH=000lB，…，9H=1001B，AH=1010B，…，FH=1111B。

8 个位(严格地说是 8 个二进制位)作为一个整体就构成 1 个字节。计算机中的存储容量就是用字节数来度量的，常见的度量单位有 KB、MB、GB 和 TB，其换算关系为

$$1KB = 2^{10}B = 1024B$$

$$1MB = 2^{10}KB = 2^{20}B = 1048576B = 1024KB$$

$$1GB = 2^{10}MB = 2^{30}B = 1073741824B = 1024MB$$

$$1TB = 2^{10}GB = 2^{40}B = 1099511627776B = 1024GB$$

ARM 体系结构中，“字”数据类型如图 1-1 所示，是将 4 个字节联合成的一个整体。因此 1 个字由 32 个二进制位(4Bytes)构成，双字就是 2 个字一起作为数据整体的二进制数，它有 64 个二进制位(8 Bytes)。

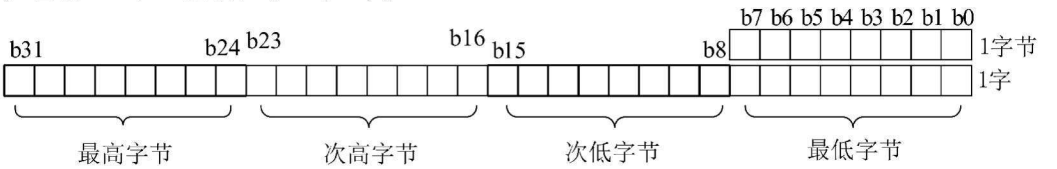

图 1-1 ARM 体系结构中的字节数据和字数据

在计算机系统中，数据的传输和存储通常以 2^n 个字节为单位，如 1 个字节、2 个字节或 4 个字节等。计算机中存储的多字节数据一定占用连续的存储器单元，各种类型的数据可以交叉存储，如字节之后存储的是双字数据，然后存储字数据等。计算机中存储的数据根据其具体应用不同，可能是数据，也可能是地址信息或状态信息等。

2. 机器数与真值

数学上用“+”和“-”表示数的正和负，在计算机中，约定用“0”和“1”分别表示符号位“+”和“-”，约定一个数的符号位始终在最左边，这个符号位的右面是各数据位，计算机中把这样的带符号二进制数称为机器数，而把用“+”和“-”表示符号的带符号数称为真值。机器数是将符号“+”和“-”也“数字化”的数，是数值在计算机中的二进制表示形式。

机器字长是指计算机中央处理单元(Central Processing Unit,CPU)一次可处理的二进制数码的位数，通常 CPU 中寄存器的数据宽度等于机器字长。机器数有两个特点：一是符号数字化；二是机器数的表示范围或表示精度受机器字长的限制。默认情况下，使用数据的最高位作为符号位，当其为“0”时，表示正数；而当其为“1”时，表示负数。这就是说，数的符号在计算机中已被数字化。

机器数除了可以写成二进制的形式外，通常也写成十六进制(H 后缀或 0x 前缀)的形式。如真值为+2754 的数所对应的机器数可写为 0101011000010B 或 0AC2H 或 0xAC2；真值为-2754 的数所对应的机器数则可写为 1101011000010B(原码)或 1AC2H 或 0x1AC2。

当用 H 作为后缀表示十六进制数时，为了区分标识符与数值，若十六进制数的最高位为 A、B、C、D、E、F 的某一数值符号时，在该数值的最前面再加一个 0，表示它是数值而非标识符。但该数值在计算机中传输和存储时，添加的 0 既不需要传输也不占用存储空间，或者说这个 0 实际上是不存在的，只是数值书写的标识而已，目的是使人们识读更方便，也为了方便编译器区分标识符和数值。譬如将十六进制数 0xF0A8 书写为 0F0A8H(编译器的主要功能是将一种高级语言翻译成低级的计算机语言，如将汇编语言程序翻译为机器语言)。

3. 原码、反码与补码

机器数的符号数字化后，带符号数在计算机中的传输、存储就没有任何问题了。但在进行运算时又应该如何处理呢？符号位需要区别对待还是与数值位一起进行处理呢？为了解决这些问题，引入了 3 种不同编码形式的机器数，即原码、反码和补码，分别用 $[X]_{原}$、$[X]_{反}$、$[X]_{补}$表示。

1) *原码*

用“0”和“1”分别表示数的符号“+”和“-”，然后紧跟数值部分(该数的绝对值的二进制表示形式)就是这个数的原码形式的机器数。例如当计算机字长为 8 位时，$[+31]_{原}=00011111B$，$[-4]_{原}=10000100B$。

在原码表示法中，根据定义，数 0 的原码有两种不同的形式，字长为 8 位时的两种表示分别为 $[+0]_{原}=00000000B$，$[-0]_{原}=10000000B$。

原码表示简单易懂，而且与真值的转换方便。但原码表示的数不便于计算机运算，因为两个原码数进行运算时，首先要判断它们的符号，然后再决定是用加法还是用减法，因而使计算部件的结构复杂化，运算时间更长。为解决上述弊病，引入了反码和补码表示法。

2）反码

对于正数，反码与原码相同，在最左边的是符号位为0(最高位)，其余位为数值位本身；对于负数，符号位为1，数值位按位求反(原来为1的位变为0,而原来为0的位变为1)。例如:$[+31]_{反}=00011111B$,$[-0]_{反}=11111111B$,$[-4]_{反}=11111011B$。

在反码表示法中,根据定义,数0的反码也有两种不同形式,字长为8位时0的两种反码表示分别是$[+0]_{反}=00000000B$,$[-0]_{反}=11111111B$。反码表示相对原码更复杂,但用反码运算时,则要简单一些,不需要判断数值的正负,也不需要比较数值的大小。但运算结果的最高进位需要返回到最低位进行修正,才能得到正确的结果(如用反码计算8+(-1)=?),从而影响了运算速度。

3）补码

对于正数，补码与原码、反码相同，即符号位为0，其余位为数值位本身；对于负数，求其补码有两种方法：(1)从其正数(绝对值)开始，写出其原码，再带符号位逐位取反，最后加1；(2)直接由原码求补码，方法是负数补码的符号位按定义应该为1，从原码数值的最右边开始向左找到第一个1，这个1及其右边的所有0都保持不变，其余各位按位求反。$[+31]_{补}=00011111B$，$[+0]_{补}=00000000B$，$[-4]_{补}=11111100B$。特别要注意的是，计算机中的机器数都是用补码表示的。表1-1是8位二进制数码表示法对照表。

表1-1　8位二进制数码表示法对照表

8位二进制数码组合	十六进制数	无符号数(十进制)	有符号数(十进制)			8位二进制数码组合	十六进制数	无符号数(十进制)	有符号数(十进制)		
			原码	反码	补码				原码	反码	补码
00000000	00H	0	+0	+0	+0	10000000	80H	128	-0	-127	-128
00000001	01H	1	+1	+1	+1	10000001	81H	129	-1	-126	-127
00000010	02H	2	+2	+2	+2	10000010	82H	130	-2	-125	-126
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
01111100	7CH	124	+124	+124	+124	11111100	FCH	252	-124	-3	-4
01111101	7DH	125	+125	+125	+125	11111101	FDH	253	-125	-2	-3
01111110	7EH	126	+126	+126	+126	11111110	FEH	254	-126	-1	-2
01111111	7FH	127	+127	+127	+127	11111111	FFH	255	-127	-0	-1

在补码表示法中，数值0的标识形式是唯一的00000000B，这一点完全不同于数值0的原码表示和反码表示。补码数11111111B的真值是-1；补码数1000000B的真值是-128。补码的运算比较简单，而且可以将减法运算变换为加法运算。因此在计算机系统中，只需要设计加法器电路，就可完成加法和减法运算，并通过循环加法实现乘法、循环减法实现除法，再进一步实现任意复杂程度的运算。不过为了使运算速度得到提高，现代计算机系统中直接设计了硬件乘法器。

由于补码具有运算过程简单、效率高、硬件电路结构简单和便于实现各种运算等优点，计算机系统中的带符号数都用补码表示。

计算机中用补码表示带符号数，如果带符号数的位数是一个字节，方便在字长为8位的计算机上处理，但是如果要在字长为16位或32位的计算机上进行数据处理，就不方便了，此时就需要在这个补码数的左边添加0或者1，使其数据宽度增加到16位或32位，

这就是补码数的符号位扩展，简称符号扩展。补码数的符号扩展方法是新增位用符号位填充，符号位是0就在前面直接补0，符号位是1就全部补1。简单地分析后可以发现，补码数的符号扩展后位数实现了扩展，但数的真值保持不变，下面举例说明。

如果8位二进制补码的最高位(符号位)为0，那么扩展后的16位补码直接在最高位前面添加8个0即可。

如果8位二进制补码的最高位(符号位)为1，那么扩展后的16位补码直接在最高位前面添加8个1即可。

真值-13的8位原码是1000 1101，8位补码是1111 0011，为了让16位计算机处理方便，需要将它扩展为16位二进制，在补码的表示形式上，最高位为1，所以直接在最高位前面添加8个1即可变为16位补码表示，即1111 1111 1111 0011，根据前面的知识，它是-13的16位补码表示。

真值+13的8位补码是0000 1101，补码的最高位(符号位)为0，那么扩展后的16位补码直接在最高位前面添加8个0，即0000 0000 0000 1101.

数的补码进行运算时，满足

$[X]_{原}=[[X]_{补}]_{补}$

$[X+Y]_{补}=[X]_{补}+[Y]_{补}$

$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$

4）*补码运算的溢出及其判断方法*

在计算机系统中，两个带符号数的补码运算的结果若超出了允许的表示范围，就会发生溢出，称为补码溢出，简称溢出。溢出即意味着结果是错误的。如何判断溢出呢？进行加法运算时，如果次高位和最高位都不产生进位或都产生进位，结果不溢出；而如果有且只有一处产生进位，则结果溢出。进行减法运算时，如果次高位和最高位都不产生借位或都产生借位，结果不溢出；而如果有且只有一处产生借位，则结果溢出。

例如，字长为8位的二进制数用补码表示时，其范围为$-2^{8-1}\sim+2^{8-1}-1$，即-128~+127，如果运算结果超出此范围，就会产生溢出。

【例1-1】 已知X=0b 01000000，Y=0b 01000001，进行补码的加法运算。

$[X]_{补}$=0b 01000000　　(+64的补码)

+ $[Y]_{补}$=0b 01000001　　(+65的补码)

即 $[X]_{补}+[Y]_{补}$=0b 10000001　　(-127的补码)

$[X+Y]_{原}$=0b 11111111　　(-127)(由上一行得到的补码结果0b100 00001，再求补码，即$[X+Y]_{原}=[[X]_{补}+[Y]_{补}]_{补}=[[X+Y]_{补}]_{补}$)

两正数相加，其结果应为正数，且为+129，但运算结果为负数(-127)，这显然是错误的。其原因是和数+129>+127，即超出了8位二进制正数所能表示的最大值，使数值部分占据了符号位的位置，产生了溢出错误。

【例1-2】 已知X=-1111111，Y=-0000010，进行补码的加法运算。

$[X]_{补}$=0b 10000001　　(-127的补码)

+ $[Y]_{补}$=0b 11111110　　(-2的补码)

$[X]_{补}+[Y]_{补}$=0b 101111111　　(+127的补码)

自动丢失的进位 ↑↑ 符号位