高等学校应用型本科“十三五”规划教材

机电 MECHATRONICS
计算机 COMPUTER
电子 ELECTRONICS

微机原理与嵌入式接口技术

主　编　刘显荣
副主编　张元涛　吴云君

西安电子科技大学出版社
http://www.xduph.com

高等学校应用型本科“十三五”规划教材

微机原理与嵌入式接口技术

主　编　刘显荣
副主编　张元涛　吴云君
参　编　汪德彪　常继彬
　　叶　文　范　苏

西安电子科技大学出版社

内容简介

本书是一本从实际应用出发，同时兼顾原理性内容的“微机原理及应用”课程教材。本书用基于ARM内核的STM32 CPU代替了传统的8086 CPU，并按照基础知识、基本原理和应用案例的结构重新构建了课程的三部分内容：微型计算机体系结构原理、汇编/C语言编程和接口应用技术。全新的教学内容与原有课程体系完全融合在一起，满足了课程的原理性与应用性的要求。

本书内容丰富，案例真实完整，主要采用C语言库函数实现，反映了现代微机技术发展的新水平和趋势，做到学以致用。本书内容包括微型计算机基础，Cortex－M3基础，Cortex－M3的指令系统与汇编程序设计，接口技术，中断技术，定时器与计数器，D/A与A/D转换技术，串行通信技术，存储器及其扩展，嵌入式接口技术。

本书可作为高等学校电气信息类专业本科生及非计算机专业研究生“微机原理及应用”或“微机原理与接口技术”课程的教材，还可以作为计算机及相关专业大专和各类嵌入式培训班的教材或参考书，对工程技术人员也有一定的指导意义和参考价值。

图书在版编目(CIP)数据

微机原理与嵌入式接口技术/刘显荣主编. —西安：西安电子科技大学出版社，2016.8
高等学校应用型本科“十三五”规划教材
ISBN 978－7－5606－4135－5

Ⅰ. ① 微… Ⅱ. ① 刘… 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. ① TP36

中国版本图书馆CIP数据核字(2016)第115987号

策划编辑 戚文艳 李惠萍
责任编辑 宁晓蓉
出版发行 西安电子科技大学出版社(西安市太白南路2号)
电 话 (029)88242885 88201467 邮 编 710071
网 址 www.xduph.com 电子邮箱 xdupfxb001@163.com
经 销 新华书店
印刷单位 陕西天意印务有限责任公司
版 次 2016年8月第1版 2016年8月第1次印刷
开 本 787毫米×1092毫米 1/16 印张 20.5
字 数 484千字
印 数 1～3000册
定 价 37.00元
ISBN 978－7－5606－4135－5/TP

XDUP 4427001－1

序

15 亿颗 STM32 出货，见证了变革和创新！

自 2007 年发布第一款以 ARM Cortex－M3 为内核的 32 位微控制器以来，STM32 以其宽广的产品线、无缝的兼容性、卓越的性能和日益完善的生态系统获得众多客户的青睐，引领了微控制领域的变革，推动了微控制器市场向 32 位快速转变。

近年来物联网技术飞速进步，有力推动了智慧型社会的发展。现在诸多智能设备已经从概念走向产品，渗入生活已经指日可待。物联网领域给嵌入式系统带来更加广阔的应用空间的同时也给微控制器厂商带来巨大挑战。

物联网处于摸索和成长阶段，其范畴内的应用存在多样性甚至不确定性，和传统应用比较，其复杂度更高。微控制器作为节点到设备端的核心器件，将面临更多的挑战，这些挑战体现在微控制器产品的适应性、底层软件的可靠性、上层软件的灵活性、整体方案的可靠性等诸多方面。

为迎接物联网带来的挑战，意法半导体在全力发展自身产品线的同时，全方位地布局、拓展 STM32 生态系统，积极和众多合作伙伴推出基于 STM32 的软件、方案、模块，帮助最终客户加快项目开发、缩短上市时间。实践证明，STM32 生态化策略得到了合作伙伴和用户的积极响应，在物联网的先行应用如可穿戴设备、传感器融合、移动支付等领域已经获得巨大的成功。

创新的应用需要具有创造力的人才，具有创造力的人才需要了解和掌握最新科技和产品。自推出 STM32 以来，意法半导体也一直在积极推动 STM32 进校园计划，让在校学生能够接触、实践时代最新产品，帮助学生加快适应市场对人才的需求。

喜闻重庆科技学院多名具有丰富理论教学及工程实践经验的专家学者精心编写了以 STM32 为基础的《微机原理与嵌入式接口技术》教材，我作为业界的专业人士非常欣喜，对他们富有创新性的劳动表示钦佩。此书既注重了微机理论教学，又创新地加入了工程实例，并配套了深圳市优易特科技开发有限公司专为此研发的开发板及实验箱。本人由衷地相信，本教材将为在校学生深入学习和掌握微机原理及最新的微控制器技术提供巨大的帮助。

值此 STM32 全球累计出货超过 15 亿颗的喜庆时刻，我愿同本书全体编著成员共同分享此刻愉悦激动的心情，也十分感谢你们为此书编写付出的辛勤劳动，祝贺本书成功出版！

意法半导体(ST)中国区
微控制器高级市场经理
曹锦东
2016 年 3 月

西安电子科技大学出版社

高等学校应用型本科“十三五”规划教材

编审专家委员会名单

主　　任：鲍吉龙（宁波工程学院副院长、教授）

副 主 任：彭　军（重庆科技学院电气与信息工程学院院长、教授）

张国云（湖南理工学院信息与通信工程学院院长、教授）

刘黎明（南阳理工学院软件学院院长、教授）

庞兴华（南阳理工学院机械与汽车工程学院副院长、教授）

电子与通信组

组　　长：彭　军（兼）

张国云（兼）

成　　员：（成员按姓氏笔画排列）

王天宝（成都信息工程学院通信学院院长、教授）

安　鹏（宁波工程学院电子与信息工程学院副院长、副教授）

朱清慧（南阳理工学院电子与电气工程学院副院长、教授）

沈汉鑫（厦门理工学院光电与通信工程学院副院长、副教授）

苏世栋（运城学院物理与电子工程系副主任、副教授）

杨光松（集美大学信息工程学院副院长、教授）

钮王杰（运城学院机电工程系副主任、副教授）

唐德东（重庆科技学院电气与信息工程学院副院长、教授）

谢　东（重庆科技学院电气与信息工程学院自动化系主任、教授）

湛腾西（湖南理工学院信息与通信工程学院教授）

楼建明（宁波工程学院电子与信息工程学院副院长、副教授）

计算机大组

组　　长：刘黎明（兼）

成　　员：（成员按姓氏笔画排列）

刘克成（南阳理工学院计算机学院院长、教授）

毕如田（山西农业大学资源环境学院副院长、教授）

李富忠（山西农业大学软件学院院长、教授）

向　毅（重庆科技学院电气与信息工程学院院长助理、教授）

张晓民（南阳理工学院软件学院副院长、副教授）

何明星（西华大学数学与计算机学院院长、教授）

范剑波（宁波工程学院理学院副院长、教授）

赵润林（山西运城学院计算机科学与技术系副主任、副教授）

雷　亮（重庆科技学院电气与信息工程学院计算机系主任、副教授）

黑新宏（西安理工大学计算机学院副院长、教授）

前　言

多年来，高等学校电气信息类专业的“微机原理及应用”或“微机原理与接口技术”课程由于采用 Intel 8086/8088 CPU 及其外围芯片作为主要教学内容而始终受到业界的诟病，有人质问“8255、8279 现在还买得到吗？为什么学生还要学习那些早已淘汰的器件？”，实际上许多一线教师在上课时，也会遇到学生问“老师，这门课学了有什么用？”，我们应该如何回答这些问题呢？是不是回答他们“学生需要打下一个良好的基础，学习 8086 还是有用的，能学到许多的基础知识和基本概念”，或者是“学生的基础还不具备，只能先学习 8086，之后才能进一步深入”。随着时间的推移，这样的回答显得越来越苍白无力了。作为一门专业技术课程，讲几个概念和基础知识就可以了吗？大学四年要一直打基础吗？既然还需要以后重新学习和深入，那么这门课程的价值是不是太低了点？有多少学生有机会再去进一步深入？这样学出来的学生，会有哪家公司感兴趣？

我们既不能责怪业界人士不了解教学，也不能责怪学生太务实。想一下 8086 是 20 世纪 70 年代末推出的芯片，距今快 40 年了，它真的太老了。当前有大量的微机新技术、新器件、新应用等待学生去学习和掌握。当然，老师们看重的原理、基础确实也很重要，不过这些基础性的内容并非 8086 的专利，当代 CPU 中所包含的原理和基础知识只会比 8086 更加全面和更加深刻，是时候改变了。

针对这种情况，本书以 ST 公司的 STM32 F1xx CPU 为核心介绍了微处理器体系结构、指令系统及汇编/C 语言编程、接口应用技术三部分内容。在大幅度更新内容的同时，保持了课程结构的稳定。ST 公司的 STM32 F1xx CPU 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学习活动；有较低的应用成本和不是太高的学习门槛，有利于实践训练和快速入门。

本书既重视接口技术的实际应用，又重视基本概念和基本原理，二者协调一致，相得益彰。在协调好汇编语言与 C 语言矛盾的基础上，本书主要采用 C 语言库函数实现各例程。在介绍中断原理时，就引入了 STM32 的外部中断应用，原理与应用一同剖析。在总结了各类微机体系接口扩展原理和实际应用要求之后，充分利用 STM32 的独特优势，将系统扩展技术归纳成四种方法：利用片内资源扩展、I/O 模拟时序接口扩展、时序可编程芯片扩展(FSMC 总线)和通信扩展。此外，还打破了传统的先主机、再外设的教学顺序，将存储器部分安排在本书靠后的位置，以期学生尽快接触接口应用技术，感受到微型计算机的魅力。

需要说明的是，本书并没有太多的手册性内容，也没有给出各种寄存器位定义的详细解释。我们认为，教材既没有办法，也没有必要体现 STM32 CPU 方方面面的技术细节，在基本原理、方法和应用技术讲清之后，一些细节资料性的东西，读者还是要去查阅相应的数据手册或电子化的帮助文档。

全书分 10 章，第 1 章主要介绍微型计算机的基础知识、基本组成和工作原理；第 2 章

介绍了 CPU 的内核——Cortex - M3 的基础知识；第 3 章介绍了 Cortex - M3 的指令系统与汇编程序设计方法；第 4 章介绍了接口的基本工作原理，以及 STM32 的 GPIO 构成与应用；第 5 章介绍了中断的基础知识和 STM32 外部中断的构成与应用；第 6 章介绍了定时器与计数器的基本工作原理和 STM32 定时器溢出、输入捕捉、输出比较、PWM 输出等内容；第 7 章介绍了 D/A、A/D 转换原理和 STM32 的 DAC 和 ADC 应用；第 8 章介绍了串行通信的基本原理和 STM32 USART 的应用；第 9 章介绍了存储器的工作原理以及存储器进行扩展的基本方法；第 10 章介绍了接口扩展的四种方法，并给出了应用案例。

本书可作为高等院校电子信息工程、通信工程、电子科学与技术、电子工程、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。若学时紧张，可不讲第 9 章和第 10 章的部分内容。

参加本书编写的教师多年来一直从事“微机原理及应用”课程的理论与实践教学，同时也有一定的项目开发经验。本书由刘显荣组织编写，第 1、4 章由汪德彪编写，第 2、5 章由常继彬编写，第 3、8 章由张元涛编写，第 6 章由叶文编写，第 7 章由吴云君编写，第 9 章由刘显荣和范苏共同编写，第 10 章由刘显荣编写。

重庆科技学院胡文金教授和汪德彪教授对本书的编写提出了宝贵的建设性意见，汪德彪教授还审阅了全稿，叶文老师、吴云君老师和张元涛老师修正了本书的一些错误。

本书配套实验板的设计与制作得到了深圳市优易特科技开发有限公司的大力协助，重庆科技学院自动化专业 2014 级的谭锐、李军美、魏俊、易斌、王奘、谢馨宇、何开美同学为本书绘制了大量的插图，在此表示衷心的感谢。

本书有配套的实验板可供实践，感兴趣的读者朋友可访问 https://u-easytech.taobao.com，查找“EDU - STM32 开发板”购买。本书的程序均通过实际调试，可以正常运行，但由于篇幅限制，书上并没有给出全部的代码，相关代码随实验板一同提供。除书中的程序以外，实验板上还提供了多个综合案例程序，供读者深入学习。读者也可加入本书书友 QQ 群 45639366，进行技术交流及资料下载。

由于作者水平有限，书中不足之处在所难免，期望能起到抛砖引玉的作用。

敬请读者批评指正。

编　者

2016 年 2 月

目　录

第 1 章　微型计算机基础

本章要点

☆ 计算机中的数制、编码及其运算
☆ 微型计算机、嵌入式系统及 ARM 的发展历程
☆ 微型计算机的体系结构及其相关概念
☆ STM32 系列微处理器介绍

本章首先关注的是用于计算的数和用于显示、传输的编码，它们分别抽象地表示了现实世界中的信息，是计算机信息加工的基本元素。结合嵌入式应用，本章介绍了算术、逻辑运算规则，对于补码和各种运算结果的特征标志应 多加留意。在兼顾微型计算机结构、工作过程等传统知识的同时，本章引入了哈佛结构、RISC、指令流水线等新概念。在介绍计算机技术发展历程的时候，本章还体现了封闭与共享、标准与创新的矛盾，以及各种理念与技术交叉融合的发展趋势，值得大家深入思考。

1.1　计算机中的数制与编码

现代计算机都是数字计算机，其内部信息都是用二进制表示的，它可以分为两类：一是用于控制计算机操作的命令信息即指令代码，二是用于处理和运算的数据信息。有的数据信息表示数量的大小和符号，有确定的数值；有的信息则没有确定的数值大小关系，比如字符、汉字、逻辑数据和图像数据等。本节介绍数制及其运算和常用的符号编码。

1.1.1　数制及其转换

1. 进位计数制

进位计数制是人们最常用的计数方法。比如计时用的六十进制(1 小时＝60 分钟)，计年用的十二进制(1 年＝12 月)等。在日常生活中最常用的是十进制，但计算机中只使用二进制。进位计数制用式(1－1)表示。

$$N=\sum_{i=-m}^{n} d_i R^i \tag{1-1}$$

式中，数字 N 有 m 位小数、$n+1$ 位整数；d_i 是数位 i 的取值，是几进制数就有几种取值可能，比如十进制有 0～9 十种取值，十六进制有 0～9、A～F 十六种取值，二进制只有 0、1 两种取值；R^i 是数位 i 的权，其中 R 为基数，二进制 $R=2$，十进制 $R=10$，十六进制则有 $R=16$。例如：

$$(110101.1)_2=1\times2^5+1\times2^4+0\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}$$

$$(148.9)_{10}=1\times10^2+4\times10^1+8\times10^0+9\times10^{-1}$$

$$(AE.8)_{16}=A\times16^1+E\times16^0+8\times16^{-1}$$

为了区分不同进制数据，在书写时通常需要标注，比如在前面的表示中，在圆括号外

以下标表示进制，也可以用英文符号下标或后缀表示不同的进制，B 表示二进制(Binary)，D 表示十进制(Decimal)，H 表示十六进制(Hexadecimal)，Q 表示八进制(Octal)。需要注意的是，如果没有特别标识，本书默认为十进制数。

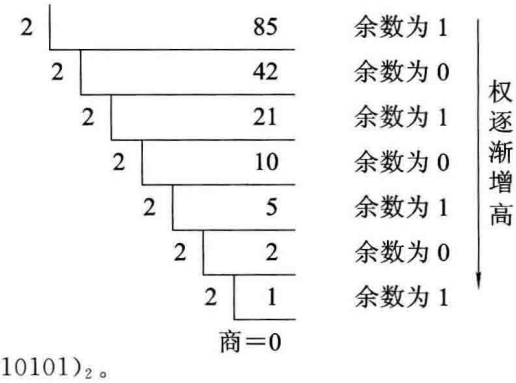
2. 不同进制之间的转换

1) 二进制与十进制之间的转换

(1) 十进制整数转换成二进制数。

除 2 取余法：十进制整数除以 2 取余，再将商除以 2，再得余数，直到商为 0 为止；先得的余数权低，后得的余数权高。

【例 1.1】 将十进制整数$(85)_{10}$采用除 2 取余法转换成二进制数，其过程如下：



得到的二进制数为$(01010101)_2$。

权重法：从低位到高位，二进制各位的权分别为 1、2、4、8、16、32、64、128、…，因此对于 255 以内的十进制数还可以采用权重法进行快速转换。

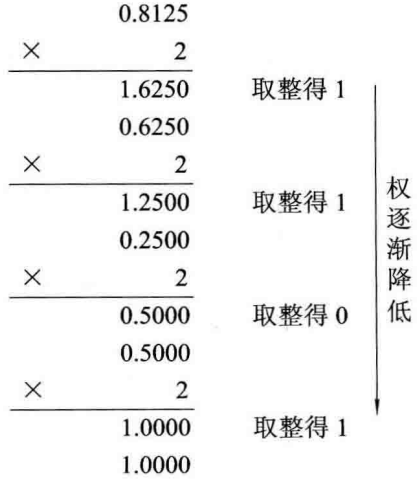
【例 1.2】 将十进制整数$(85)_{10}$采用权重法转换成二进制数，其过程如下：

$(85)_{10}=(64)_{10}+(16)_{10}+(4)_{10}+(1)_{10}=(01010101)_2$

(2) 十进制小数转换成二进制数。

乘 2 取整法：将十进制小数乘以 2 取整，取整后的小数再乘以 2 取整，直到取整后的小数为 0 或满足精度要求为止；先得的整数权高，后得的整数权低。

【例 1.3】 将十进制小数$(0.8125)_{10}$转换成二进制数，其过程如下：



得到的二进制数为$(0.1101)_2$。

(3) 二进制数转换成十进制数。

无论是整数还是小数，将二进制数转换成十进制数根据式(1-1)按权展开求和即可。

【例 1.4】 将二进制数$(1011.01)_2$转换成十进制数，其过程如下：

$$(1011.01)_2 = 1\times2^3+0\times2^2+1\times2^1+1\times2^0+0\times2^{-1}+1\times2^{-2}$$
$$=8+0+2+1+0+0.25=(11.25)_{10}$$

2) 十进制与十六进制、八进制之间的转换

十进制转换成十六进制跟十进制转换成二进制的方法类似。对于整数的转换是将十进制数除以 16 取余，直到余数为 0 为止，先得的余数权低，后得的余数权高。对于小数的转换是将十进制数乘以 16 取整，取整后小数再乘以 16 取整，直到取整后的小数为 0 或满足精度要求为止，先得的整数权高，后得的整数权低。

【例 1.5】 将十进制数$(75.375)_{10}$转换成十六进制数，其过程如下：

整数部分：
```
16 | 75    余数为 11
   | 16 | 4    余数为 4
   |    | 商=0
```

小数部分：
```
  0.375
×   16
-------
 6.000    取整得 6
```

在十六进制中，需要 16 个符号表示数位，十进制的 0～9 与十六进制的 0～9 完全相同，十进制的 10～15 对应十六进制的 A～F，所以最后结果为

$$(75.375)_{10}=(4B.6)_{16}$$

反过来，将十六进制转换成十进制，其方法依然是按权展开求和。

【例 1.6】 将十六进制数$(2CF.8)_{16}$转换成十进制数，其过程如下：

$$(2CF.8)_{16}=2\times16^2+12\times16^1+15\times16^0+8\times16^{-1}=719.5$$

同样的道理，十进制与八进制之间的转换也可以如法炮制，在此不赘述。

3) 二进制与十六进制、八进制之间的转换

(1) 二进制与十六进制之间的转换。

二进制转换成十六进制以小数点为界，对于整数部分，从最右边开始，将 4 位二进制数划为一个整体，不足 4 位时在左边添 0 补足 4 位；对于小数部分，从最左边开始，将 4 位二进制数划为一个整体，不足 4 位时在右边添 0 补足 4 位；然后按照表 1.1 的对应关系，直接写出转换结果。

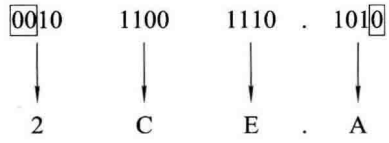
表 1.1　十、二、八、十六进制数对应关系表

十进制	二进制	八进制	十六进制
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5

续表

十进制	二进制	八进制	十六进制
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

【例 1.7】 将二进制数$(1011001110.101)_2$转换成十六进制数，其过程如下：

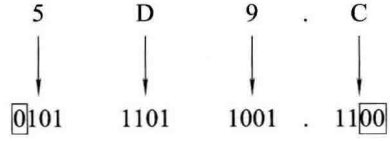


所以，转换结果为

$$(1011001110.101)_2=(2CE.A)_{16}$$

反过来，将十六进制转换成二进制时，用 4 位二进制数去取代 1 位十六进制数，将整数部分最左边和小数部分最右边的 0 去掉即可。

【例 1.8】 将十六进制数$(5D9.C)_{16}$转换成二进制数，其过程如下：



所以，转换结果为

$$(5D9.C)_{16}=(10111011001.11)_2$$

(2) 二进制与八进制之间的转换。

二进制与八进制之间的转换跟二进制与十六进制之间的转换相似，不同的是 3 位二进制数对应 1 位八进制数，其转换过程不再赘述。

1.1.2 计算机中数的表示与运算

众所周知，数字计算机只能接受数字信息，任何信息都必须用二进制编码来表示。数值数据信息分为无符号数(Unsigned Number)和有符号数(Signed Number)。数字信息中所有的二进制位都表示数值大小的数称为无符号数；数字信息中除了表示数值大小的数位外，还用 1 位二进制位表示数的符号，这就是有符号数。符号位通常处于数的最高位，用“0”表示正，用“1”表示负。

1. 二进制无符号数的运算

1）加法运算

运算法则：0 + 0 = 0；1 + 0 = 1；0 + 1 = 1；1 + 1 = 0(产生进位)。

【例 1.9】 求两个 8 位无符号二进制数 1011_0110B(182D)与 0110_0101B(101D)的和。

$$\begin{array}{rr} & 10110110 \\ + & 01100101 \\ \hline & \boxed{1}\,00011011 \end{array} \quad C_y=1$$

如果加法器只有 8 位，它只能存留 8 位运算结果，相加结果的最高位则无法保留在最终结果中，而是保留在进位(Carry)标志里，进位标志通常用 C_y 表示。显然，182+101=283，而运算结果为 27，与 283 相差了 256，进位 $C_y=1$ 即表示 256。

2）减法运算

运算法则：1−0 = 1；0−0 = 0；1−1 = 0；0−1 = 1(产生借位)。

【例 1.10】 求两个 8 位无符号二进制数 0111_0110B(118D)与 1000_1101B(141D)的差。

$$\begin{array}{rr} & 01110110 \\ - & 10001101 \\ \hline & \boxed{1}\,11101001 \end{array} \quad B_w=1$$

与加法运算一样，如果运算器是 8 位的，则只能存留 8 位运算结果。在本例中，118 不够 141 减，但运算结果为 233，显然结果不正确，难道计算错了？这是因为无符号数运算不可能产生负数，118 与 141 相差 23，不够减而产生一个借位 $B_w=1$，这个借位的值是 256，扣除相差的 23 后，正是运算结果 233。

在计算机中进位和借位在物理上是同一个地方，通常把进位 C_y 和借位 B_w 统称为 C 标志，在加法运算时它是进位标志，在减法运算时它是借位标志。

3）乘法运算

运算法则：0×0=0；0×1=0；1×0=0；1×1=1。

【例 1.11】 求两个二进制数 1010B(10D)与 0110B(6D)的乘积。

$$\begin{array}{rr} & 1010 \\ \times & 0110 \\ \hline & 0000 \\ & 1010 \\ & 1010 \\ + & 0000 \\ \hline & 0111100 \end{array}$$

从本例看出，两个 4 位二进制数相乘的结果是 8 位二进制数。一般地，一个 n 位二进制数乘以 m 位二进制数，乘积不超过 $n+m$ 位。

要实现乘法运算过程并不是一件容易的事，在计算机中实现乘法运算一般有三种方法。

(1) 软件编程实现。在早期的计算机中，没有乘法电路，只能通过编程方法用移位运算和加法运算来实现，比如被乘数左移法、部分积右移法等。

(2) 加法器＋硬件辅助电路实现。在加法器和移位寄存器的基础上，再增加必要的辅

助电路实现乘法运算，比如 Intel 8086 就是采用这种方法。

(3) 专用乘法器实现。在 CPU 中除了加法器和移位寄存器外，专设硬件高速乘法部件，专门负责乘法运算，运算效率高。现代的 CPU 均有专门的乘法部件。

4) 除法运算

除法运算也是通过移位、减法等运算联合实现的。与乘法运算类似，实现除法运算一般也有三种方法：一是完全通过软件编程实现；二是通过减法器＋硬件辅助电路实现；三是用专用除法器实现。

2. 二进制有符号数的表示与运算

二进制有符号数表示形式如下：

符号	尾数

符号：用 1 个 bit(位)表示，“0”表示“＋”，“1”表示“－”。

有符号数有三种表示方法：原码、反码和补码，其中补码最为重要和最为常用。

1) 原码

数值最左边一位为符号位，用 0 表示正数，用 1 表示负数，其余的位表示数值的大小。假设有符号数 X，它的原码表示形式为$[X]_{原}$。

例如，假设用 8 位二进制数表示，则

$$[+37]_{原}=0\_0100101B,\ [-37]_{原}=1\_0100101B$$

如果一个有符号数用 n 位二进制数表示，除去 1 位符号位外，用于表示数值的位数为 $n-1$ 位，则 n 位二进制数原码能表示有符号数的范围为$-2^{n-1}+1\leqslant X\leqslant+2^{n-1}-1$。

2) 反码

数值最左边一位为符号位，用 0 表示正数，用 1 表示负数。对于正数，其反码与原码相同；对于负数，数值位逐位取反，即数值为 1 的位变成 0，而数值为 0 的位变成 1。假设有符号数 X，它的反码表示形式为$[X]_{反}$。

例如，假设用 8 位二进制表示，则

$$[+37]_{反}=0\_0100101B\ ,\ [-37]_{反}=1\_1011010B$$

如果一个有符号数用 n 位二进制数表示，则反码能表示数的范围为$-2^{n-1}+1\leqslant X\leqslant+2^{n-1}-1$。

3) 补码

数值最左边一位为符号位，用 0 表示正数，用 1 表示负数。对于正数，其补码与原码相同；负数则在其反码的基础上加 1。假设有符号数 X，它的补码表示形式为$[X]_{补}$。

例如，假设用 8 位二进制表示，则

$$[+37]_{补}=0010\_0101B\ ,\ [-37]_{补}=1101\_1011B$$

如果$[+37]_{补}+[-37]_{补}=0010_0101B+1101_1011B=1_0000_0000$，即低 8 位全为零，向前有一进位，说明二者互为补，1_0000_0000 则是全集。

如果一个有符号数用 n 位二进制数表示，则其补码能表示数的范围为$-2^{n-1}\leqslant$

$X \leqslant +2^{n-1}-1$。

4）补码数运算

有符号数引入补码后，可以将加法运算和减法运算统一为加法运算，并能实现数值位与符号位一起运算，自动得出运算结果。运算规则如下：

$$[X]_{补}+[Y]_{补}=[X+Y]_{补}$$

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

$$[-X]_{补}=\overline{[X]}_{补}+1$$

【例 1.12】 用补码计算 +29+18=？

假设用 8 位二进制数表示，则

$$[+29]_{补}=00011101\text{B},\ [+18]_{补}=00010010\text{B}$$

$$\begin{array}{rll} & [+29]_{补} & 00011101 \\ + & [+18]_{补} & 00010010 \\ \hline & [+47]_{补} & 00101111 \end{array}$$

自动得到正确的运算结果 +47。

【例 1.13】 用补码计算 +29−18=？

假设用 8 位二进制数表示，则可以理解为减去一个数等于加上这个数的补码。

$$[+29]_{补}=00011101\text{B}$$

$$[+18]_{补}=00010010\text{B},\ [-18]_{补}=11101110\text{B}$$

$$\begin{array}{rll} & [+29]_{补} & 00011101 \\ + & [-18]_{补} & 11101110 \\ \hline & [+11]_{补} & \boxed{1}\ 00001011 \end{array} \qquad C_y=1$$

虽然在运算过程中产生了进位，但还是得到正确结果 +11，进位自动丢失。

可以利用常见的时钟方法来理解补码的运算规则。如图 1.1 所示，当前时针指向 7 点，如要将其拨到 2 点，可以逆时针拨 5 点(−5)，也可以顺时针拨 7 点(+7)，对于时钟而言，7 就是 5 的补码，减去 5 就等于加上 7，二者取得了相同的结果。需要注意的是时钟是 12 进制的，在加 7 的时候，时针越过 12 点，相当于有一个进位，这个进位在本次补码运算中被丢弃了。

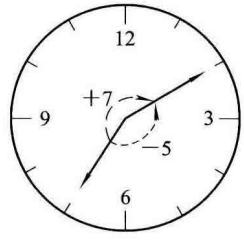


图 1.1　补码的时钟图

3. 运算溢出及其判断

前面的例子说明有符号数补码运算能自动得出结果，即使产生了进位，其结果也是正确的。试问每次运算的结果都是正确的吗？答案是否定的，先看一个例子。