普通高等教育“十一五”国家级规划教材

高等学校计算机硬件技术课程系列教材

微机原理与接口技术(第2版)

谢瑞和　等编著

高等教育出版社
Higher Education Press

普通高等教育“十一五”国家级规划教材

高等学校计算机硬件技术课程系列教材

微机原理与接口技术

（第2版）

谢瑞和　等编著

高等教育出版社

内容提要

本书以 Pentium 系列微处理器为主要背景，全面讲述了微型计算机的组成原理与接口技术。首先简述了微处理器的基本结构、CPU 寄存器与 PC 主板；重点介绍了 IA 指令系统、汇编语言程序设计技术与开发调试方法；接着详细描述了存储器及其接口技术，Cache 技术，基本 I/O 接口技术，中断技术，可编程接口芯片及其应用，键盘、显示器、ADC 与 DAC 等应用接口电路设计，LPT 并行接口，COM 串行接口，USB 接口，ISA 扩展总线与 PCI 扩展总线；最后介绍了保护模式下段页式管理的实现原理及其增强 PC 性能的一系列新技术。

本书集编者从事微机技术应用教学与科研 20 多年的学识，以及编写出版 10 余本计算机类图书之经验，在内容选材、层次分布、概念描述、科学性与实用性等方面颇具融合力。

本书可作为高等院校理工类相关专业“微机原理与接口技术”课程的教材，也可作为工程设计与科技人员的参考用书。

图书在版编目(CIP)数据

微机原理与接口技术／谢瑞和等编著. —2 版. —北京:高等教育出版社,2007.7

ISBN 978-7-04-021958-6

Ⅰ.微… Ⅱ.谢… Ⅲ.①微型计算机—理论—高等学校—教材 ②微型计算机—接口—高等学校—教材

Ⅳ.TP36

中国版本图书馆 CIP 数据核字(2007)第 090889 号

策划编辑 孙惠丽　**责任编辑** 孙　薇　**封面设计** 于文燕　**责任绘图** 朱　静

版式设计 陆瑞红　**责任校对** 王效珍　**责任印制** 韩　刚

出版发行 高等教育出版社

社　　址 北京市西城区德外大街 4 号

邮政编码 100011

总　　机 010—58581000

经　　销 蓝色畅想图书发行有限公司

印　　刷 北京民族印刷厂

开　　本 787×1092　1/16

印　　张 18.75

字　　数 450 000

购书热线 010—58581118

免费咨询 800—810—0598

网　　址 http://www.hep.edu.cn
http://www.hep.com.cn

网上订购 http://www.landraco.com
http://www.landraco.com.cn

畅想教育 http://www.widedu.com

版　　次 2004 年 8 月第 1 版
2007 年 7 月第 2 版

印　　次 2007 年 7 月第 1 次印刷

定　　价 23.60 元

本书如有缺页、倒页、脱页等质量问题，请到所购图书销售部门联系调换。

物料号 21958-00

第2版前言

本书第1版《32位微型计算机原理与接口技术》于2004年7月出版后，编者通过3届本科生的教学实践，并在广泛征求校外各方面意见的基础上，对其进行了全面修订。经专家评审，本书被列入普通高等教育“十一五”国家级规划教材。这使我们感到无比荣幸，然而也深知责任之重大。

同初版相比，本书具有以下几个显著的特点。

① 加强了基础部分，尤其在汇编语言程序设计与I/O接口设计这两个关键部分充实了大量的应用设计实例，包括汇编语言程序开发与调试方面的相关知识，增强学生对基础知识的理解，提高学生应用设计的能力，使之更好地适应本科生专业基础课程的教学要求。

② 增强了传统的可编程接口芯片、ADC与DAC芯片的介绍及应用设计实例，作为I/O接口技术的重点，以便同当前较为流行的接口技术实验装置相匹配。然而，也保留了用LPT标准实施I/O接口设计的特色内容，扩展了I/O接口技术的应用范例，有的学校或读者还能以此为参考，方便而经济地利用LPT资源做I/O实验。

③ 将初版“接口篇”的并行接口、串行接口、PCI总线与USB接口等章节压缩成第8章。将初版“提高篇”压缩成第9章，并给这两章与第4章的4.9节、第5章的5.3节与5.4节标识*号。如果学时不够，可以将其中的有些内容留给学生自修，本版对这些章节做了精心的组织与表述，力求在教师不讲授的前提下，便于学生自学理解，吸引读者了解当代微机发展新技术的基本原理，拓展知识面与解决问题的思路。

④ 鉴于64位微机已全面推向市场，以32位微处理器为背景组织微机原理与接口技术的教学已成为共识，故本书书名已没有必要再冠以“32位”。

本书由谢瑞和、左冬红、杨明、翁虹等编写，参加编写的还有董毅、张士军、马爱梅、王红、林卫、程世平等多年主讲微机原理与接口技术以及实验课的教师。国防科技大学邹逢兴教授审阅了书稿并提出了宝贵意见；高等教育出版社有关领导与编辑为本书的出版付出了辛勤的劳动；华中科技大学校系领导为本书的编写创造了良好的条件，在此一并致谢。

由于编者个人学识有限，书中难免会有错误或不妥之处，诚挚期待同行与读者批评赐教。来信请寄华中科技大学电子与信息工程系谢瑞和，邮编430074，电子邮箱xieruihe@public.wh.hb.cn。

编著者

2007年4月

第1版前言

每当我们主讲微型计算机原理与接口技术课程时，不时有学生发问："我们现在使用的计算机都是 Pentium 3 或 Pentium 4，为什么学的还是 20 世纪 70 年代的 8086、8088？"。面对这类问题，作为教师确实没有任何充足的理由能说服他们。

接口技术也是一样，学的是 ISA 扩展总线与 825x 系列早期接口芯片，但使用的都是 PCI 扩展总线和功能强大的超大规模芯片组，或者各类可编程逻辑器件，时差也有约 20 年。

在和许多兄弟院校的学习交流中，大家都一致认为应该彻底改变这种现象，但难于找到一本真正适合本科教学的 32 位微型计算机原理与接口技术教材。

本书试图在这方面作一些尝试，全书以 Pentium 系列微处理器为主线，介绍 32 位微型计算机原理与接口技术。在内容的选材与安排上颇具特色，彻底抛弃了同类教材中近 20 年来基本维持不变的传统内容，实际上这些内容早已被当代微型计算机所淘汰。根据由浅入深的原则，精心组织内容，将其划分为基础、接口与提高三个层次，不仅能明显提高学生学习效率，而且显著压缩了全书的篇幅。基础篇主要讲述 Pentium 系列微处理器的基本结构与中断技术等基本操作原理、指令系统和汇编语言程序设计。接口篇以崭新的面貌区别于传统的同类教材，首先系统介绍了 Pentium 微处理器的总线操作与时序、存储器与 I/O 接口技术，然后分章详细讨论了当代 PC 的串行接口、并行接口、USB 接口与 PCI 扩展总线。提高篇首先全面剖析了 Pentium 系列的操作模式，尤其对保护模式进行了系统而精辟的描述，然后详细介绍了 Pentium 系列的 MMX、超标量、动态执行、分支预测、条件传送及 CPU 识别等一系列增强技术以及 Pentium 系列主板与芯片组。

本书集编著者 20 多年从事微型计算机技术应用的教学、科研与写作经验，在内容层次、语言表达以及概念描述等诸多方面都力争便于读者自学。本书是高等院校理工专业微型计算机原理与接口技术类课程的通用教材。同时我们正在着手编著同本书配套的实验课教材及开发相应实验装置，不久即可问世。

参加本书编著的有谢瑞和、翁虹、张士军、杨明、马爱梅、曾延安等多年主讲微型计算机原理与接口技术的教师。全书由谢瑞和主编、统一策划。第 1～4 章由谢瑞和、翁虹、张士军、杨明等集体编著，第 5～15 章由谢瑞和编著。周丽军、张金与何博等研究生为本书的例题程序和接口技术实例做了大量开发工作。高等教育出版社领导与编辑以非凡的决策与胆略全力支持本书出版，在此一并致谢。

由于我们组织 32 位微型计算机原理与接口技术教学的时间还不长，积累的经验不多，编著者个人学识很有限，书中难免会有不少错误或不妥之处，诚心期待同行与读者批评赐教。

编 者

2004 年 4 月于华中科技大学

目 录

第1章 计算机运算基础

计算机是用来进行各种数据运算与信息处理的工具，尽管这些被处理的信息千差万别，但它们都是以二进制数据的形式来操作的。本章简要地概述了计算机中使用的数制系统及其几种常用的编码。许多读者对这些内容并不陌生，但作为计算机的基础知识来温习一遍，有助于更好地理解计算机操作的机理，提高学习后续章节的效率。

1.1 无符号数

无论计算机如何发展，它的内部操作总是基于由“0”与“1”组成的二进制数，换言之，计算机实质上只能识别出二进制的“0”和“1”，只是由不同的二进制位的排列与组合可以编制出各种不同的复杂操作。

1.1.1 二进制与十六进制

1. 二进制数

二进制数的基数为2，逢二进一，它的多项式表示为

$$b_n\times 2^n+b_{n-1}\times 2^{n-1}+\cdots+b_1\times 2^1+b_0\times 2^0+b_{-1}\times 2^{-1}+b_{-2}\times 2^{-2}+\cdots$$

二进制整数部分的位权从小到大依次为 2^0、2^1、2^2、2^3、2^4、2^5、2^6、2^7、…，亦即为十进制的1、2、4、8、16、32、64、128、256、…。二进制小数部分的位权从大至小依次为 2^{-1}、2^{-2}、2^{-3}、2^{-4}、…，亦即为十进制的1/2、1/4、1/8、1/16、…。各位的系数只有“0”与“1”两种选择，例如

$$(1101.11)_2=1\times 2^3+1\times 2^2+0\times 2^1+1\times 2^0+1\times 2^{-1}+1\times 2^{-2}$$

可见它等于十进制的13.75。

科技文献中规定二进制数以英文字母“B”（Binary）为后缀标记，如1101.11B。

在二进制中经常使用K（Kilo）、M（Mega）、G（Giga）等作为计量单位，应注意它们同十进制表示的数有所差异。

$$1\ \mathrm{K}=2^{10}=1\ 024$$

$$1\ \mathrm{M}=2^{20}=1\ 048\ 576$$

$$1\ \mathrm{G}=2^{30}=1\ 073\ 741\ 824$$

2. 十六进制数

十六进制数的基数是16，逢十六进一，它的多项式表示为

$$h_n\times16^n+h_{n-1}\times16^{n-1}+\cdots+h_1\times16^1+h_0\times16^0+h_{-1}\times16^{-1}+h_{-2}\times16^{-2}+\cdots$$

十六进制整数部分的位权从小到大依次为16^0、16^1、16^2、16^3、16^4、…，亦即十进制的1、16、256、4 096、65 536、…。十六进制小数部分的位权从大到小依次为16^{-1}、16^{-2}、16^{-3}、16^{-4}、…，亦即十进制的1/16、1/256、1/4 096、1/65 536、…。它的系数有16种数字，前10个为十进制数字0～9，后六个为英文字母A、B、C、D、E、F或者它们对应的小写字母，分别表示十进制数的10、11、12、13、14、15。例如，

$$(89AB.4)_{16}=8\times16^3+9\times16^2+10\times16^1+11\times16^0+4\times16^{-1}$$

可见它等于十进制的35 243.25。

科技文献中规定十六进制数以英文字母“H”(Hexadecimal)为后缀标记，如89AB.4H。必须注意的是，当十六进制数的最高位为A～F中的任何一个字母数字时，为了与非数字的字符串相区别，规定应在最前面加一位数字“0”。例如，十六进制数ABCD3456H是非法的书写格式，而应改写为0ABCD3456H。当最高位为数字0～9时，则默认为是数据而非字符串。例如，十六进制数9FEDCH是合法的书写法。

顺便说明，在微型计算机的数据表示格式中，不带后缀标记或者带后缀“D”的数将默认为十进制数，例如$(254)_{10}$记为254D或254。

表1.1列举了部分二进制数、十进制数与十六进制数。这里所有的位都用来表示数值，所以是无正负符号位的数。

表1.1 无符号二进制、十进制和十六进制的对照

二进制	十进制	十六进制	二进制	十进制	十六进制	二进制	十进制	十六进制
00000000B	0	00H	00001000B	8	08H	00010000B	16	10H
00000001B	1	01H	00001001B	9	09H	00010001B	17	11H
00000010B	2	02H	00001010B	10	0AH	00010010B	18	12H
00000011B	3	03H	00001011B	11	0BH	00010011B	19	13H
00000100B	4	04H	00001100B	12	0CH	00010100B	20	14H
00000101B	5	05H	00001101B	13	0DH	00010101B	21	15H
00000110B	6	06H	00001110B	14	0EH	00010110B	22	16H
00000111B	7	07H	00001111B	15	0FH	00010111B	23	17H

在计算机中将8位二进制数称为一个字节（Byte），数据的存储与处理往往以字节为基本单元。两个字节组成的16位二进制数称为一个字（Word），4个字节组成的32位二进制数称为一个双字（Two Word），8个字节组成的64位二进制数称为一个四字（Quit Word）。由此可见：

对于8位无符号的二进制数，能够表示的十进制数范围是0～255；

对于16位无符号的二进制数，能够表示的十进制数范围是0～65 535；

对于32位无符号的二进制数，能够表示的十进制数范围是0～4 294 967 295。

1.1.2 数制转换

在上述二进制数与十六进制数的介绍中已经描述了将它们转换成十进制数的方法，这种方

法也适用于其他任何非十进制数，亦即只需将非十进制数以多项式的形式展开，然后计算它们的十进制数之和即可将其转换为十进制数。

下面讨论十进制数转换成非十进制数以及二进制数同十六进制数之间的相互转换。

1. 十进制数转换为非十进制数

将十进制数整数部分转换成其他进制，只需将该十进制数除以该进制的基数，将所得的商数再除以基数，直至不能整除为止，然后取每次相除所得的余数，按“后高前低”的顺序排列即为转换结果。

将十进制数小数部分转换成其他进制，则应将该进制的基数乘以十进制数小数部分，取出乘积的小数部分再乘基数直至乘积为 0，或结果达到预定的精度，然后将每次乘积的整数按“前高后低”的顺序排列即为转换结果。

下面讨论几种常见的转换。

① 十进制数转换为二进制数，采用除 2 取余法，即将十进制整数除以 2，得到一个商数和余数，再将商数除以 2 又得到一个商数和余数，如此反复除下去直至不能整除为止。以最后所得的商数“1”作为最高位，将各次所得的余数按“后高前低”的顺序写下来即得用二进制表示的结果。

【例 1.1】 100=(?)B。

转换过程如图 1.1 所示。

除数	被除数/商数	余数	
2	100		
2	50	0	↑ 最低位
2	25	0	
2	12	1	
2	6	0	
2	3	0	
	1	1	次高位

图 1.1 除 2 取余法

结果得 100=1100100B。

② 同样的方法可将十进制数转换成十六进制数，采用除 16 取余法，即将十进制整数除以 16，得到一个商数和余数，再将商数除以 16 又得到一个商数和余数，如此继续除下去直至不能整除为止。以最后所得的商数作为最高位，将各次所得的余数按“后高前低”的顺序写下来即得用十六进制表示的结果。

【例 1.2】 35 243=(?)H。

转换过程如图 1.2 所示。

结果得 35 243=89ABH。

③ 将十进制数小数部分转换成二进制小数，采用乘 2 取整法，即将十进制纯小数部分乘 2，摘除乘积中的整数后保留小数部分再乘 2，如此继续下去直至乘积小数部分为零或者得到要求的位数为止。将各次获取的整数依“前高后低”的顺序写出来即为转换后的二进制纯小数结果。

除数	被除数/商数	余数	
16	35243		↑最低位
16	2202	11	
16	137	10	
	8	9	次高位

图 1.2 除 16 取余法

【例 1.3】 0.1875=(?)B。

转换过程如图 1.3 所示。

	积的整数部分	被乘数/积的小数部分	乘数
最高位↓		0.1875	2
	0	0.375	2
	0	0.75	2
	1	0.5	2
最低位	1	0.0	

图 1.3 乘 2 取整法

结果得 0.1875=0.0011B。

④ 将十进制数小数部分转换为十六进制小数，采用乘 16 取整法，即将十进制纯小数部分乘 16，摘除乘积中的整数后保留小数部分再乘 16，如此继续下去直至乘积小数部分为零或者得到要求的位数为止。将各次摘取的整数依“前高后低”的顺序写出来即为转换后的十六进制纯小数结果。

【例 1.4】 0.78125=(?)H。

转换过程如图 1.4 所示。

	积的整数部分	被乘数/积的小数部分	乘数
最高位↓		0.78125	16
	12	0.5	16
最低位	8	0.0	

图 1.4 乘 16 取整法

结果得 0.78125=0.C8H。

2. 十六进制数与二进制数之间的转换

由于 1 位十六进制数实际上是 4 位二进制数，所以两者之间的转换是轻而易举的。将二进制数转换成十六进制数时，以小数点为界，整数部分自右至左每 4 位一组，最高位部分不足 4 位时在左边补 0，每组用对应的一位十六进制数表示；而小数部分则自左至右每 4 位一组，最低位部分不足 4 位时在右边补 0，每组用对应的一位十六进制数表示。例如：

10 1101 1010 0011.0110 11 B
=2 D A 3. 6 CH

注意，不要将十六进制数小数点最后一位“C”误写为“3”。

由此可见，引入十六进制数的主要目的是为了免除书写与阅读一长串二进制数码的麻烦，克服容易出错之弊病，计算机本身并不需要做转换运算。

如果要将十六进制数转换成二进制数，只需将十六进制数的每一位用对应的 4 位二进制数表示，并依原顺序排列即可。例如：

5 F 8 4. E 4 H
= 0101 1111 1000 0100.1110 0100 B

1.2 带符号数

如同用“+”、“–”符号表示正数与负数一样，当数据带符号时，计算机用数据的最高位作为符号位，且规定该位为“0”表示正数，该位为“1”表示负数。这样，在一个字节数据的 8 位二进制中就只有 7 位表示数值了。同理，在 16 位二进制的字数据中就只有 15 位表示数值；32 位的双字数据中则只有 31 个数据位。由于这里将符号位数字化，因此带符号数同无符号数两者在表达形式上看不出任何差别，但它们所表示的数值范围是完全不同的。

1.2.1 负数的表示法

引入符号位之后，由于正数的符号位为 0，同不带符号的数相比并没有发生“质”的变化，只是可以表示的数值范围变小了。

例如不带符号时，8 位二进制数 01111111B 表示 127，11111111B 表示 255，它是可以表示的最大数。考虑符号位之后，+127 依然用 01111111B 来表示，然而它却成了 8 位二进制中可以表示的最大正数，因为再加 1 即成为 10000000B，数值位向符号位进位，结果是负数了，运算出错，此种现象称为溢出。

那么负数又如何表示呢？下面以–72 为例来说明它的表示方法与步骤：

① 写出该负数对应的正数的二进制数，例如 72=01001000B。

② 将该二进制数按位取反，即 1 改写为 0，0 改写为 1，这个过程简称为“取反”(Not)，相当于逻辑**非**，例如 01001000B 取反得 10110111B。

③ 再在最低位加上 1，例如 10110111B 加 1 后得 10111000B。

整个过程称为“求补”(Complement)，其结果就是负数的补码格式，因此 - 72=B8H。

由于构成计算机的数字逻辑电路对于“取反”、“加 1”、“进位”等操作轻而易举，所以引入补码表示负数为计算机的运算操作提供了极大的方便，提高了机器的运算效率，例如采用这种格式表示负数后，减法就可以当作加法来操作。

【例 1.5】 127–16 =?

因为 127–16=127+(–16)，而–16 的补码格式表示为

```
    0  0  0  1  0  0  0  0
    1  1  1  0  1  1  1  1   ;取反
+                   1       ;加1
=   1  1  1  1  0  0  0  0
```

所以计算机只需做以下加法：

```
    0  1  1  1  1  1  1  1
+   1  1  1  1  0  0  0  0
= 1 0  1  1  0  1  1  1  1
```

忽略进位即为运算结果，用十六进制表示是6FH。将其转换为十进制来验证：

01101111B=6FH=6×16+15=111

1.2.2 符号数的运算

因为任何正数加上它对应的负数必为0，所以它们互补。对正数求补可得到它对应的负数，对负数求补又可以得到对应的正数，也就是得到了绝对值。因此带符号数的换算并不复杂。

已知二进制的符号数，将它转换为十进制的方法如下：

① 判断最高位，即符号位是0，还是1。如果符号位是0，说明是正数，此时直接转换即可。

② 如果符号位是1，说明是负数，对该数求补。

③ 将所得结果转换成十进制数，该数对应的负数即为所求结果。

【例1.6】 将符号数88H转换成十进制数。

88H=10001000B，符号位为1是负数，因此要对88H求补：

```
    1  0  0  0  1  0  0  0
    0  1  1  1  0  1  1  1        ；取反
+                   1            ；加1
=   0  1  1  1  1  0  0  0 = 78H
```

因为78H=120，故结果得出符号数88H=–120。

表1.2列出了部分带符号数与不带符号数的对应关系，从中可以发现它们拥有以下一些特点：

① 8位带符号数的数值范围为$-2^7 \sim +2^7-1$ (–128～+127)。

② 16位带符号数的数值范围为$-2^{15} \sim +2^{15}-1$ (–32 768～+32 767)。

③ 32位带符号数的数值范围为$-2^{31} \sim +2^{31}-1$ (–2 147 483 648～+2 147 483 647)。

④ 当要将8位带符号数扩展成16位带符号数，或者将16位带符号数扩展成32位带符号数时，只需将最左边的符号位再往左延伸至16位或者32位即可。

【例1.7】 将8位带符号数64H(100)扩展为16位带符号的二进制数。

64H=01100100B，根据符号位向左扩展的法则，用16位带符号二进制表示则是

00000000 01100100B=0064H

【例1.8】 将8位带符号数88H扩展为16位带符号的二进制数。

表 1.2 8 位/16 位/32 位二进制无符号/带符号数的表示法

8 位			16 位			32 位		
十六进制	无符号十进制	带符号十进制	十六进制	无符号十进制	带符号十进制	十六进制	无符号十进制	带符号十进制
00H	0	0	0000H	0	0	00000000H	0	0
01H	1	+1	0001H	1	+1	00000001H	1	+1
…	…	…	…	…	…	…	…	…
7FH	127	+127	7FFFH	32 767	+32 767	7FFFFFFFH	2 147 483 647	+2 147 483 647
80H	128	–128	8000H	32 768	–32 768	80000000H	2 147 483 648	–2 147 483 648
81H	129	–127	8001H	32 769	–32 767	80000001H	2 147 483 649	–2 147 483 647
…	…	…	…	…	…	…	…	…
0FEH	254	–2	0FFFEH	65 534	–2	0FFFFFFFEH	4 294 967 294	–2
0FFH	255	–1	0FFFFH	65 535	–1	0FFFFFFFFH	4 294 967 295	–1

88H=10001000B，根据符号位向左扩展的法则，用 16 位带符号二进制表示则是

11111111 10001000B=0FF88H

带符号数的运算方法同不带符号数的运算方法相同，运算结果倘若发生了进位或借位，而且如果忽略了它们的影响，那么其结果必然是错误的。但是带符号数的运算除考虑进位或借位外，还必须注意溢出的问题。下面举几个带符号数的运算实例。

【例 1.9】 01000000B+00100011B=01100011B。

这里两个正数相加的结果为正数，运算有效。

【例 1.10】 01111111B+00000010B=10000001B。

这里两个正数相加的结果为负数，产生了溢出，即两数之和超出了 8 位符号数所能表示的最大值+127，结果出错。

【例 1.11】 10000001B+10001111B=C+00010000B。

这里两个负数相加的结果产生了进位，使得和的 8 位符号数为正数，结果出错。

【例 1.12】 01000001B–10101011B=01000001B+01010101B=10010110B。

这里是一个正数减去一个负数，等效于该正数同这个负数的补码相加，由于运算结果产生溢出，使得结果出错。

【例 1.13】 1234H+4FEDH=6221H。

此例是两个正数相加，结果为正数，既无进位也未溢出，运算有效。

【例 1.14】 0A9876543H–12345678H=97530ECBH。

此例是一个负数减去一个正数，结果为负数，既无进位也未溢出，运算有效。

值得指出的是，由以上的讨论绝不能得出以下错误的结论，即认为 8 位计算机只能处理–128～+127 之间的数据，而 16 位计算机只能作–32 768～+32 767 之间的运算。上面讨论的 8 位/16 位/32 位是指计算机单次操作的运算长度，计算机能够由多次操作来处理超出微处理器位长的大量程数据。从理论上讲，其数值范围的大小几乎没有什么限制。当然位数多的计算机，例如 32 位微型计算机，一次运算操作就相当于 8 位/16 位微型计算机的多次操作，从而大大提高了操作效率与数据处理能力。

1.3 运算方法

计算机中常用的数据运算方法如表 1.3 所示。算术运算和逻辑运算用在传统的计算机技术中，不必赘述。这里引入了饱和运算这一个新的概念，下面举几个例子做一点简要的说明。

表 1.3 计算机常用运算方法

分 类	运 算	以字节举例	说 明
算术运算	加法	29H+A8H=D1H	结果超出原数据类型范围时产生数值的溢出、进位或借位，置运算结果的相应状态位为 1。用于数值计算
	减法	F8H–19H=DFH	
	乘法	20H×03H=60H	
	除法	66H÷06H=11H	
饱和运算	加法	85H+7FH=FFH	超出原数据类型范围时不产生数值的进位、借位与溢出，将结果限定在最大值和最小值的范围内，常用于多媒体数据的处理
	减法	AAH–98H=12H	
	乘法	10H×0FH=F0H	
逻辑运算	或	(10011011B)+(11000011B)=11011011B	按位进行逻辑运算处理,常用于逻辑控制类场合
	与	(10011011B) · (11000011B)=10000011B	
	异或	(10011011B) ⊕ (11000011B)=01011000B	

【例 1.15】 16 位运算时结果出现进位的实例。

1234H+0FEDCH=11110H，显然，运算结果超出了 16 位，进位加至第 17 位，如果计算机运算字长只有 16 位，则进位的“1”实际上会丢失，此时运算结果的状态寄存器将设置一个产生了进位的标志，以供下一步作相应处理。

饱和运算则不同，它将运算结果限定在指定的范围内，结果超出最大值后仍取最大值，不产生最高位进位，结果小于最小值仍取最小值，不产生最高位借位。

【例 1.16】 24 位饱和算法的实例之一。

778899H+885533H=0FFDDCCH，这里两个和数较小，没有产生进位现象。

【例 1.17】 24 位饱和算法的实例之二。

708899H+905400H=0FFFFFFH。若按算术运算处理，此加法结果应为 100DC99H，然而第 25 位的 1 会丢失，实际结果为 00DC99H。可以想象，倘若它们是图像的亮度数据，则这两个信号相加后会出现黑白倒置的现象；如果是两个较强的声音信号，则合成后的声音会变得很弱。这就是在声音、图像等多媒体数据处理中通常采用饱和算法的原因。Pentium 系列的 MMX 指令集就是基于这种算法。

1.4 常用的编码

计算机总是以量化的数据形式来进行操作的，但它处理的对象不一定都是“数”，例如现