普通高等教育“十一五”国家级规划教材
中国科学院电子信息与通信系列规划教材

微机原理与接口技术

楼顺天　周佳社　编著

科学出版社

北　京

内 容 简 介

本书以 Intel 公司生产的 8086/8088 CPU 为核心，详细介绍汇编语言程序设计技术、系统总线形成、存储器设计、常用和专用芯片的接口技术及其应用编程方法。

在汇编语言程序设计中，分别介绍计算机中的数制和码制、补码的运算规则、数据和转移地址的寻址方式、8086/8088 的指令系统，着重介绍汇编语言的编程技术，并结合示例介绍许多实际应用编程技巧，强调汇编语言中指针的使用。在接口技术中，介绍 8086/8088 系统总线的形成、常用芯片与系统总线的接口、专用芯片的接口与工作方式控制、中断技术及其应用，重点介绍存储器的设计和专用芯片的应用设计，结合示例介绍一些实际应用系统的设计方法。

本书可作为高等院校各专业本科生的教材，也可供相关工程技术人员、管理人员和自学者参考。

图书在版编目(CIP)数据

微机原理与接口技术/楼顺天，周佳社编著. —北京：科学出版社，2006
（普通高等教育“十一五”国家级规划教材·中国科学院电子信息与通信系列规划教材）
ISBN 7-03-017566-2

Ⅰ.微… Ⅱ.①楼… ②周… 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(2006)第 072704 号

责任编辑：匡 敏 余 江 潘继敏 / 责任校对：邹慧卿
责任印制：张克忠 / 封面设计：陈 敬

科学出版社 出版
北京东黄城根北街 16 号
邮政编码：100717
http://www.sciencep.com

中国科学院印刷厂印刷

科学出版社发行 各地新华书店经销

*

2006 年 8 月第 一 版 开本：B5(720×1000)
2006 年 8 月第一次印刷 印张：25 1/4
印数：1—4 000 字数：494 000

定价：28.00 元

（如有印装质量问题，我社负责调换〈环伟〉）

《中国科学院电子信息与通信系列规划教材》
编 委 会

丛　书　序

信息技术的高速发展及广泛应用，使信息技术成为当今国际竞争中最重要的战略技术。信息技术对经济建设、社会变革乃至国家安全起着关键性的作用，它是经济发展的“倍增器”和社会进步的“催化剂”，是体现综合国力的重要标志。在人类历史上，没有一种技术像信息技术这样引起社会如此广泛、深刻的变革。在20世纪末和21世纪前半叶，信息技术乃是社会发展最重要的技术驱动力，可以说，21世纪人类已经步入了信息时代。信息产业在世界范围内正在由先导产业逐步变为主导产业。从微观上看，表现为单位产品的价格构成中，能源和材料的消耗减少而信息技术和信息服务的比重上升；从宏观上看，表现为国民生产总值（GDP）中信息产业所占的比重增加。一个国家信息产业的发展水平将是衡量该国社会经济总体发展和现代化程度的重要标志之一。

目前，信息科学已成为世界各国最优先发展的科学之一。党的十六大提出了“加速发展信息产业，大力推进信息化，以信息化带动工业化”的发展战略，以及“优先发展信息产业，在经济和社会领域广泛应用信息技术”的基本国策，使我国信息产业得到了前所未有的重视，信息产业呈现出飞速发展的势头。信息产业的发展离不开信息化人才，信息化人才建设将是信息产业可持续发展的关键。然而，有关调查表明，我国国家信息化指数为38.46，而信息化人才资源指数仅为13.43。据权威机构预测，从2005年到2009年，中国信息行业将以18.5%的年复合增长率高速增长，中国信息市场将迎来又一个“黄金年代”。

为了适应新世纪信息学科尤其是电子信息与通信学科的长足发展，在规模上、素质上更好地满足我国信息产业和信息科学技术的发展需要，更好地实现电子信息与通信学科专业人才的培养目标，推进国内信息产业的发展，中国科学院教材建设专家委员会和科学出版社组织电子信息与通信领域的院士、专家、教学指导委员会成员、国家级教学名师及电子信息与通信学科院校的相关领导等组成编委会，共同组织编写这套《中国科学院电子信息与通信系列规划教材》。

本套教材主要面向全国范围内综合性院校电子信息工程、通信工程、信息工程等相关专业的本科生。本套教材的编委会成员具有国内电子信息与通信方面的较高学术水平，他们负责对本套教材的编写大纲及内容进行审定，可使本套教材的质量得以保证。

本套教材主要有以下几方面的特点：

1. 适应多层次的需要。依据最新专业规范，系列教材主要根据教育部最新公布的电子信息与通信学科相关专业的“学科专业规范”和“基础课程教学基本要求”

进行教材内容的安排与设置。同时，根据各类型高校学生的实际需要，编写不同层次的教材。

2. 结构体系完备。本套教材覆盖本科、研究生教学层次，各门课程的知识点之间相互衔接，以便完整掌握学科基本概念、基本理论，了解学科整体发展趋势。

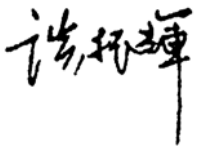
3. 作者水平较高。我们将邀请设有电子、通信国家重点学科的院校，以及国家级、省级教学名师或国家级、省级精品课程负责人编写教材。

4. 借鉴国外优秀教材。编委会为每门课程推荐一本国外相关的经典原版教材，作为教师编写的参考书。

5. 理论与实际相结合，加强实践教学。教材编写注重案例和实践环节，着力于学生实际动手能力的培养。

6. 教材形式多样。本套教材除主教材外，还配套有辅导书、教师参考书、多媒体课件、习题库及网络课程等。

根据电子信息与通信学科专业发展的战略要求，我们将对本套系列教材不断更新，以保持教材的先进性和适用性。热忱欢迎全国同行以及关注电子信息与通信领域教育及教材建设的广大有识之士对我们的工作提出宝贵意见和建议。

北京交通大学校长 

2005 年 10 月

前　言

在大多数高等院校的本科生教学中，计算机课程的设置和教学实施得到了广泛重视，一般都开设了“计算机文化基础”、“C 语言程序设计”、“软件技术基础”、“微机原理与接口技术”（或“微机原理与系统设计”）等课程，本书就是专门为“微机原理与接口技术”（或“微机原理与系统设计”）课程所编写的本科生教材，适合各个专业使用。

微机原理与接口技术课程是重要的专业基础课，是学生参加全国、省三级计算机等级考试（PC 技术）的重要课程，也是电路、信号与系统、信号处理、自动控制等硕士学科的考研科目之一。这门课程是“计算机文化基础”，“C 语言程序设计”的继续，又是“数据结构”、“软件技术基础”、“计算机网络”等课程的基础，在计算机教学中占据重要地位。

本书作者均有十几年“微机原理与系统设计”课程的教学经验，多年来一直承担着研究生试卷的命题任务，完成了十多项相关的科研项目，自主开发了“8086 单板型微型计算机实验教学系统”和“基于 EISA/PCI 系统总线的微机原理及接口技术实验教学系统”，并有出版 10 多本教材和专著的写作经验。本书具有下列特点：

（1）从学生学习、教师讲授的角度出发，内容由浅入深，循序渐进，前后连贯并呼应，使结构系统而完整。

（2）内容层次分明，既有基础知识的系统介绍，又有拓宽知识的基本介绍，给学生留下广阔的思维空间。

（3）理论联系实际，在知识介绍的同时，结合实际示例，采用面向应用的启发式教学方法。

（4）以编程思路为主线，介绍汇编语言的程序设计方法，让学生切实掌握编程知识。

（5）以实际应用设计为主线，介绍微机系统的接口设计技术。

（6）将工程设计中常用的实例、常见问题解决方法等作较全面的总结，提高了本书的实用价值。

（7）写作语言流畅，内容选择合理，结构编排适当。

本书共分 11 章，由楼顺天、周佳社共同编写，楼顺天负责统稿。其中第 1～4章、第 9～10 章由楼顺天编写，第 5～8 章和第 11 章由周佳社编写。

第 1～4 章构成 8086 汇编语言程序设计技术，重点介绍 8086 的指令系统及其编程方法。第 5～11 章构成了 8086 的系统总线及其接口设计技术，重点介绍 8086/8088 系统的总线形成、存储器设计、常用芯片接口设计、专用芯片

(8259、8253、8255）的接口设计、实际应用接口（D/A、A/D、非编码矩阵键盘、LED数码显示、光电隔离、步进电机等）的设计及其应用编程。

为方便教师授课和学生学习，附录A列出8086/8088的指令系统，附录B给出DOS中断INT 21H的部分功能列表，附录C是本书的例题索引。

感谢西安电子科技大学的有关老师给予本书写作的支持，感谢参与本书绘图和校稿的研究生，更要感谢科学出版社的编辑给予细致的润色加工及发行同志为本书的推广所作的辛苦努力。

由于作者水平和知识面的限制，书中难免会有错误和欠妥之处，敬请读者批评指正，以便及时更正，使本书更有益于广大的读者。

编　者

2006年5月

目　　录

第 1 章　数制与码制

1.1　数制表示及其转换

1.1.1　数制的表示

因为人有 10 根手指，所以自古以来就习惯使用 10 根手指来计数，因此逢十进一的十进制系统很自然就成为人类常用的计数方法。

数制是以表示数值所用的数字位数来命名，例如，十进制用 10 位数字（0～9）表示，二进制用 2 位（数字 0、1）表示，十六进制用 10 位数字和 6 位符号（A、B、C、D、E、F）表示。各种数制中数字或符号的个数称为数制的基数。

任意进制数可以通过多项式形式表示，设数制的基数为 b，则数 x 可以表示成

$$x=\sum_{i=-m}^{n}k_i b^i=k_{-m}b^{-m}+\cdots+k_{-1}b^{-1}+k_0+k_1 b+\cdots+k_n b^n \qquad (1.1)$$

其中 k_{-m}，…，$k_n\in[0,1,\cdots,b-1]$，m、n 为非负整数。上式表示数 x 可以表示成 b 进制数，整数 $n+1$ 位，小数 m 位。这一式子也称为数值的按权值表示。

（1）十进制。

常用的十进制数可以直接用 0～9 数字表示，也可以在数字后加 D（decimal）表示，例如，257 和 369.2D 可以按式（1.1）分别表示成

$$257=2\times10^2+5\times10+7\times10^0$$

$$369.2\mathrm{D}=3\times10^2+6\times10+9\times10^0+2\times10^{-1}$$

（2）二进制。

在数字计算机中，经常用二进制数来表示数值，这是因为在数字电路中，只能用高电平和低电平表示不同的事件。二进制数可以用 0～1 数字后加 B（binary）表示，例如，10101B 可以按权值展开成

$$10101\mathrm{B}=1\times2^4+0\times2^3+1\times2^2+0\times2^1+1\times2^0$$

这里介绍几个常用的术语：

位（bit）：二进制位，只有两种状态：0 和 1，它是计算机中存储信息的最小单位。

字节（byte）：8 个二进制位，可以存储 8 位二进制数，如果是无符号数，

则其范围为0～255。通常计算机中的存储单元按字节设置，也就是说，8086微机系统中可以访问的最小存储单元为一个字节，1 byte＝8 bit。

字（word)：16个二进制位，2个字节，可以存储16位二进制数，如果是无符号数，则其范围为0～65535。

双字（double word)：32个二进制位，2个字，4个字节，可以存储32位二进制数，如果是无符号数，则其范围为0～4294967295。

字长：基本数据单元所包含的二进制位数，8086微处理器中经常采用的字长为8和16。

(3) 十六进制。

为书写表示方便，通常将4位二进制数看作为1位十六进制数，这时用数字0～9和符号A～F表示，并在十六进制数后加H（heximal）表示。在书写十六进制数时，如果最高位是字符，则在其前面要加上0，以便与标识符区别开来。这样我们有

0AH＝1010B　　0BH＝1011B　　0CH＝1100B

0DH＝1101B　　0EH＝1110B　　0FH＝1111B

十六进制数也可以表示成权值展开形式，如

$$325\mathrm{H}=3\times16^{2}+2\times16+5\times16^{0}$$

$$0\mathrm{B6H}=11\times16+6\times16^{0}$$

1.1.2　数制的转换

同一个数值可以用各种数制来表示，各种数制表示的数值之间可以进行转换。

1. 十进制数转换成其他进制数

将十进制数 N 分成两部分：整数部分 z 和纯小数部分 f。设要将十进制数 $z.f$ 转换成 b 进制数，则整数部分 z 采用除 b 取余的方法，即有

$$\begin{aligned} z_1 &= \left[\frac{z}{b}\right} \quad & y_1 \\ z_2 &= \left[\frac{z_1}{b}\right] & y_2 \\ &\vdots & \\ z_n &= \left[\frac{z_{n-1}}{b}\right] & y_n \end{aligned}$$

其中 z_1，…，z_n 为商，y_1，…，y_n 为余数，[·] 表示取整运算。当 $z_n=0$ 时迭代过程终止，这样得到的 y_1，…，y_n 就是 b 进制数的各位数字，y_1 为最低位，y_n 为最高位。

例 1.1　将十进制数125转换成二进制数。

解：转换过程为

$$
\begin{array}{r|l}
2 & 125 \\
\hline
2 & 62 \quad \cdots\cdots 1 \\
\hline
2 & 31 \quad \cdots\cdots 0 \\
\hline
2 & 15 \quad \cdots\cdots 1 \\
\hline
2 & 7 \quad \cdots\cdots 1 \\
\hline
2 & 3 \quad \cdots\cdots 1 \\
\hline
2 & 1 \quad \cdots\cdots 1 \\
\hline
 & 0 \quad \cdots\cdots 1
\end{array}
$$

因此，125＝1111101B。

纯小数部分 f 采用乘以 b 取整的方法，即有

$$
\begin{aligned}
r_1 &= [f \times b] & f_1 &= f \times b - r_1 \\
r_2 &= [f_1 \times b] & f_2 &= f_1 \times b - r_2 \\
&\vdots \\
r_m &= [f_{m-1} \times b] & f_m &= f_{m-1} \times b - r_m
\end{aligned}
$$

其中 r_1，…，r_m 为取整结果，f_1，…，f_m 为小数部分。当 $f_m=0$ 时迭代过程终止，这样得到的 r_1，…，r_m 就是 b 进制数的各位数字，r_1 为最高位，r_m 为最低位。

例 1.2　将十进制数 0.6875 转换成二进制数。

解：转换过程为

$$
\begin{array}{r}
0.6875 \\
\times \quad 2 \\
\hline
1.3750 \quad \cdots\cdots 1 \\
0.375 \\
\times \quad 2 \\
\hline
0.750 \quad \cdots\cdots 0 \\
\times \quad 2 \\
\hline
1.50 \quad \cdots\cdots 1 \\
0.5 \\
\times \quad 2 \\
\hline
1.0 \quad \cdots\cdots 1
\end{array}
$$

因此，0.8125＝0.1011B。

2. 其他进制数转换成十进制数

将任意进制数变换成十进制数，可以按照权值表示进行展开，例如

$$\begin{aligned}10110110\text{B} &= 1\times2^{7}+1\times2^{5}+1\times2^{4}+1\times2^{2}+1\times2^{1}\\ &= 128+32+16+4+2=182\\ 1011.011\text{B} &= 1\times2^{3}+1\times2^{1}+1\times2^{0}+1\times2^{-2}+1\times2^{-3}\\ &= 8+2+1+0.25+0.125=11.375\\ 78.\text{AH} &= 7\times16^{1}+8\times16^{0}+10\times16^{-1}\\ &= 120.625\end{aligned}$$

1.2 二进制数的运算规则

1.2.1 二进制数的算术运算

十进制数的运算规则是我们所熟悉的。计算机中是以二进制数来表示的，为书写方便，经常写成十六进制数。因此这里主要讨论二进制数和十六进制数的算术运算规则。其他进制数的运算规则与十进制数类似，二进制数加法运算采用逢二进一，减法运算采用借一作二；十六进制数加法运算采用逢十六进一，减法运算采用借一作十六，在乘除法运算时，也采用类似的规则。例如

$$\begin{aligned}&1011011\text{B}+10011\text{B}=1101110\text{B}\\ &1011\text{B}\times10011\text{B}=11010001\text{B}\\ &65\text{H}+7\text{AH}=0\text{DFH}\\ &65\text{H}\times7\text{AH}=3022\text{H}\end{aligned}$$

1.2.2 二进制数的逻辑运算

二进制数的逻辑运算是位对位的运算，即本位运算结果不会对其他位产生任何影响，这一点与算术运算是截然不同的。二进制数的逻辑运算有四种：与（AND）、或（OR）、异或（XOR）、非（NOT），其规则如表 1.1 所示。

表 1.1 二进制数位的逻辑运算规则

输入 2 个位值	(0, 0)	(0, 1)	(1, 0)	(1, 1)
AND 运算	0	0	0	1
OR 运算	0	1	1	1
XOR 运算	0	1	1	0
NOT 运算	NOT 0=1		NOT 1=0	

例如

10010111B AND 00111000B＝00010000B

10010111B OR 00111000B＝10111111B

10010111B XOR 00111000B＝10101111B

利用逻辑运算可以完成特定的操作，AND 运算可以对指定位进行清 0，OR 运算可以对指定位进行置 1，XOR 运算可以对指定位进行取反。例如，对 x 的第 0、3 位清零操作：x AND 11110110B，对 x 的第 1、2 位置 1 操作：x OR 00000110B，对 x 的第 3、7 位取反操作：x XOR 10001000B。

1.3 有符号数的表示

利用二进制数来表示有符号数时，必须有一位用来表示符号位，一般采用最高位表示，如图 1.1 所示。这样表示的数称为机器数，其实际值称为机器数的真值。

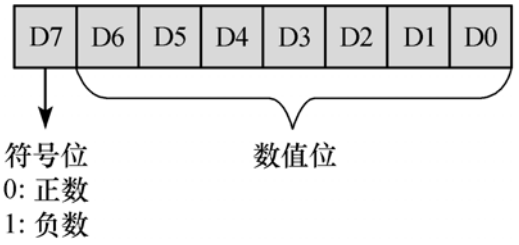


图 1.1 有符号数的表示

1.3.1 原码表示法

除符号位外，剩余 7 位就是真值的绝对位，这种表示方法称为原码表示法。例如，＋1011001B 表示成 01011001B，－1011001B 表示成 11011001B，这种表示方法的优点是直观，但加减运算时比较麻烦。例如，在对两个数进行加法运算时，应该先对其符号进行判断，如果同号，则进行相加运算，如果异号，则实际上应该进行相减运算。

另外，对于特殊值 0 有两种表示：＋0 表示成 00000000B，－0 表示成 10000000B，但实际上，00000000B 和 10000000B 表示同一个值 0。

1.3.2 补码表示法

计算机中有符号二进制数采用补码表示，x 的补码表示 $[x]_{补}$ 定义为

$$[x]_{补}=\begin{cases} x, & \text{当 } 0\leqslant x\leqslant 2^{n-1} \text{ 时,} \\ x+2^{n}, & \text{当 } -2^{n-1}\leqslant x<0 \text{ 时,} \end{cases} \quad (\mathrm{mod}2^{n})$$

求一个数 x 的补码，可以表示成 $[x]_{补}$，这种过程称为求补运算。从定义可以看出，当 x 为正数时，其原码与补码一致，只有当 x 为负数时，才有求补码的问题。

当 $n=8$ 时，有符号二进制数的表示范围为－128～127，当 $n=16$ 时，有符号二进制数的表示范围为－32768～32767。例如，＋12 的原码和补码表示均为 00001100B，而－12 的补码表示为 $[-12]_{补}=-12+2^{8}=244=$11110100B，实际上，它就是原码 10001100B 按位取反（符号位除外）再加 1 的结果。

因此负数的补码为原码取反加 1（符号位除外），即当 x 为负数时，有

$$[x]_{补} = \overline{[x]_{原}} + 1 \quad （除符号位）$$

如果对已经表示成补码的数 $[x]_{补}$ 再求补码，则可以得到其原码表示，即

$$[[x]_{补}]_{补} = [x]_{原}$$

实际上，对负数 x 求补码时，只需要先求出（$-x$）的原码，然后按位取反再加 1，即

$$[x]_{补} = \overline{[-x]_{原}} + 1 \quad （含符号位）$$

这为求负数补码运算提供了简捷的运算方法。

例 1.3 求－15 的补码表示。

解：分两步进行。

（1）$[15]_{补}$ =00001111B。

（2）$[-15]_{补} = \overline{0000\ 1111}$B+1=11110001B。

1.4 有符号数的运算及其溢出规则

1.4.1 补码运算规则

有符号二进制数以补码形式表示以后，可以直接进行加减法运算，并满足下列规则：

（1）加法 $[x+y]_{补} = [x]_{补} + [y]_{补} \quad (\mathrm{mod}2^n)$

该式表明：当带符号数的两个数采用补码形式表示时，进行加法运算可以把符号位和数值位一起进行运算（若符号位有进位，则丢掉），其结果为两数之和的补码形式。例如

(＋57)＋(＋45)= 00111001B＋00101101B = 01100110B （＋102）

(＋57)＋(－45)= 00111001B＋11010011B = [1]00001100B （进位舍弃，＋12）

(－57)＋(－45)= 11000111B＋11010011B = [1]10011010B （进位舍弃，－102）

（2）减法 $[x-y]_{补} = [x]_{补} - [y]_{补} \quad (\mathrm{mod}2^n)$

(＋57)－(＋45)= 00111001B－00101101B

= 00001100B （＋12）

(＋57)－(－45)= 00111001B－11010011B

= [1]01100110B （借位舍弃，＋102）

(－57)－(－45)= 11000111B－11010011B

= [1]11110100B （借位舍弃，－12）

（3）用加法完成相减运算 $[x-y]_{补} = [x]_{补} + [-y]_{补} \quad (\mathrm{mod}2^n)$

已知 $[y]_{补}$ 而求 $[-y]_{补}$ 的过程称为变补或求负。其规则为：对 $[y]_{补}$ 的每一位（包括符号位）进行按位取反，然后再加 1，其结果即为 $[-y]_{补}$。例如，+87 的补码表示为 01010111B，而 −87 的补码就可以这样计算：$[-87]_{补}=\overline{01010111}\text{B}+1=10101001\text{B}$。这样就又提供了一种求负数补码的方法。

（4）加法与减法互换 $[x]_{补}-[y]_{补}=[x]_{补}+[-y]_{补}\ (\mathrm{mod}2^n)$

应注意，一旦采用补码进行加减运算，所有参加运算的数及结果都是用补码表示的。计算机里的实际情况就是这样。

总结：求负数补码的三种方法：

（1）按照定义。

（2）$[x]_{补}=\overline{[x]_{原}}+1$（符号位除外）。

（3）先求 $-x$ 的补码，然后 $[x]_{补}=\overline{[-x]_{补}}+1$（含符号位）。

1.4.2 有符号数运算时的溢出问题

设计算机字长为 n 位，则有符号数的范围为

$$-2^{n-1}\leqslant x\leqslant+2^{n-1}-1$$

当 $n=8$ 时，其有符号数的范围为 $-128\leqslant x\leqslant+127$；当 $n=16$ 时，有符号数的范围为 $-32768\leqslant x\leqslant+32767$。

当两个有符号数进行加减运算时，如果运算结果超出可表示的有符号数的范围时，就会发生溢出，这时结果就会出错。溢出发生在两种情况下：两个同符号数相加，两个异符号数相减。

有符号数的溢出规则为：

（1）在加法运算时，如果次高位（数值最高位）相加形成进位，而最高位（符号位）相加（包括次高位的进位）却没有进位时，则结果溢出；或者相反，如果次高位无进位，而最高位有进位，则结果溢出。

（2）在减法运算时，如果次高位不需借位，而最高位需借位，则结果溢出；或者相反，如果次高位需借位，而最高位不需借位，则结果溢出。

在微机系统中，当加减运算溢出时，溢出标志位 OF 会自动置 1。

例 1.4 计算 (+121)+(+75) 和 (−121)−(+75)，并判断有无溢出。

解：(+121)+(+75)=01111001B+01001011B

=11000100B （−60 有溢出）

(−121)−(+75)=10000111B−01001011B=00111100B （60 有溢出）

1.5 BCD 编码方法及其运算

用 4 位二进制数来表示一位十进制数，这种方法称为 BCD 编码方法。4 位

二进制数有16种组合，如果取前10种组合分别表示0～9，这称为8421BCD码。

一般来说，8位二进制数（一个字节）可以表示两位十进制数，这种表示方法称为组合BCD数，如果8位二进制数只表示一位十进制数，则称为分离BCD数。

当计算机中两个BCD数进行运算时，由于CPU只能实现二进制数的运算，因此BCD数的运算结果可能出错。这是因为在BCD码加法运算时，应该是逢十进一，而CPU则逢十六进一（4位二进制数），因此，在某些位应作“加6修正”，BCD码加法运算的修正规则为：

（1）两个BCD码位相加无进位，并且结果少于或等于9，则该位不需要修正。

（2）两个BCD码位相加有进位，或者结果大于9，则该位应作加6修正。

（3）低BCD码位修正结果使高BCD码位大于9，则高位进行加6修正。

同样，BCD码的减法运算时，需要对BCD码位进行“减6修正”，其修正规则为：

（1）两个BCD码位相减无借位，则该位不需要修正。

（2）两个BCD码位相减有借位，则该位应作减6修正。

例1.5 已知$(56)_{BCD}$=01010110B，$(38)_{BCD}$=00111000B，计算这两个BCD数的加、减运算，并进行适当的修正。

解：$(38)_{BCD}+(56)_{BCD}$=00111000B+01010110B

=10001110B(高、低BCD码位都没有进位，

但低BCD码超出9)+0110B

=10010100B （结果为94,正确）

$(56)_{BCD}-(38)_{BCD}$=01010110B−00111000B

=00011110B(低BCD码位有借位)−0110B

=00011000B （结果为18,正确）

1.6 ASCII编码方法

在计算机中使用各种各样的程序设计语言，它们都是由字母、数字、符号构成的。而在计算机中输入、输出这些以字母、数字、符号表示的信息时，是以二进制代码表示的，因此，每个字母、数字、符号都应该对应于一个代码，这就是编码方法。

最常用的编码方法是ASCII码（American Standard Code For Information Interchange），即美国国家信息交换标准字符代码，如表1.2所示。

表 1.2　ASCII 码表

十进制值	➡	0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
⬇	十六进制值	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	空位(零位)	▶	空位(空间)	0	@	P	·	p	C	É	á	▒	└	╨	∝	≡
1	1	☺	◀	!	1	A	Q	a	q	ü	æ	í	▒	┴	╤	β	±
2	2	☻	↕	″	2	B	R	b	r	é	Æ	ó	▒	┬	╥	Γ	≥
3	3	♥	!!	#	3	C	S	c	s	â	ô	ú	|	├	╙	Π	≤
4	4	◆	¶	$	4	D	T	d	t	ä	ö	ñ	┤	—	╘	∑	⌠
5	5	♣	§	%	5	E	U	e	u	à	ò	Ñ	╡	┼	╒	O	⌡
6	6	♠	▬	&	6	F	V	f	v	ª	ō	a	╢	╞	╓	╫	÷
7	7	○	↨	′	7	G	W	g	w	ç	ù	º	╖	╟	╪	τ	≈
8	8	◘	↑	(	8	H	X	h	x	ê	ÿ	¿	╕	╚	┘	Φ	°
9	9	○	↓	)	9	I	Y	i	y	ë	Ö	⌐	╣	╔	┌	Θ	•
10	A	▩	→	*	:	J	Z	j	z	è	Ü	¬	║	╩	▐	Ω	•
11	B	♂	←	+	;	K	[	k	{	ï	¢	1/2	╗	╦	▄	δ	√
12	C	♀	∟	,	<	L	\	l	|	î	£	1/4	╝	╠	▬	∞	n
13	D	♪	↔	−	=	M	]	m	}	ì	¥	¡	╜	═	▌	∅	2
14	E	♫	▲	.	>	N	^	n	~	Ä	₧	«	╛	╬	▙	∈	■
15	F	☼	▼	/	?	O	_	o	⌂	Å	ƒ	»	┐	╧	▟	∩	空 FFH

在汇编语言程序设计中，经常要用到字符的 ASCII 码，常用的符号有：数字 0～9，英文字母 A～Z 和 a～z，空格，回车，换行，Esc 等，这些符号的 ASCII 码必须牢牢记住。

1.7　小　　结

在这一章中，从微处理器系统中经常采用的二进制数出发，介绍了十进制数转换为二进制数的方法及二进制数的运算规则，重点介绍了有符号数的两种表示方法：原码表示和补码表示，并讨论了补码运算规则、溢出规则。给出了 BCD 编码方法和 BCD 码运算的修正规则，最后给出了 ASCII 码表。

通过本章的学习，要求学生掌握数制之间的转换、有符号数的补码表示、有符号数运算时溢出判断、BCD 码运算的修正规则及常用符号的 ASCII 码。

习　题

1.1　将下列十进制数转换成二进制数：

（1）58；　　（2）67.625；　　（3）5721。

1.2　将二进制数变换成十六进制数：

(1) 10010101B; (2) 1101001011B; (3) 1111111111111101B;

(4) 0100000010101B; (5) 01111111B; (6) 010000000001B。

1.3 将十六进制数变换成二进制数和十进制数：

(1) 78H; (2) 0A6H; (3) 1000H; (4) 0FFFFH。

1.4 将下列十进制数转换成十六进制数：

(1) 39; (2) 299.34375; (3) 54.5625。

1.5 将下列二进制数转换成十进制数：

(1) 10110.101B; (2) 10010010.001B; (3) 11010.1101B。

1.6 计算（按原进制运算）：

(1) 10001101B+11010B; (2) 10111B+11100101B; (3) 1011110B-1110B;

(4) 124AH+78FH; (5) 5673H+123H; (6) 1000H-F5CH。

1.7 已知a=1011B，b=11001B，c=100110B，按二进制完成下列运算，并用十进制运算检查计算结果：

(1) a+b; (2) c-a-b; (3) a×b; (4) c÷b。

1.8 已知a=00111000B，b=11000111B，计算下列逻辑运算：

(1) a AND b; (2) a OR b; (3) a XOR b; (4) NOT a。

1.9 设机器字长为8位，写出下列各数的原码和补码：

(1) +1010101B; (2) -1010101B; (3) +1111111B;

(4) -1111111B; (5) +1000000B; (6) -1000000B。

1.10 写出下列十进制数的二进制补码表示（设机器字长为8位）：

(1) 15; (2) -1; (3) 117; (4) 0;

(5) -15; (6) 127; (7) -128; (8) 80。

1.11 设机器字长为8位，先将下列各数表示成二进制补码，然后按补码进行运算，并用十进制数运算进行检验：

(1) 87-73; (2) 87+(-73); (3) 87-(-73);

(4) (-87)+73; (5) (-87)-73; (6) (-87)-(-73)。

1.12 已知a，b，c，d为二进制补码：a=00110010B，b=01001010B，c=11101001B，d=10111010B，计算：

(1) a+b; (2) a+c; (3) c+b; (4) c+d;

(5) a-b; (6) c-a; (7) d-c; (8) a+d-c。

1.13 设下列四组为8位二进制补码表示的十六进制数，计算a+b和a-b，并判断其结果是否溢出：

(1) a=37H，b=57H; (2) a=0B7H，b=0D7H;

(3) a=0F7H，b=0D7H; (4) a=37H，b=0C7H。

1.14 求下列组合BCD数的二进制和十六进制表示形式：

(1) 3251; (2) 12907; (3) ABCD; (4) abcd。

1.15 将下列算式中的十进制数表示成组合BCD码进行运算，并用加6/减6修正其结果：

(1) 38+42; (2) 56+77; (3) 99+88; (4) 34+69;

(5) 38－42；　　(6) 77－56；　　(7) 15－76；　　(8) 89－23。

1.16　将下列字符串表示成相应的 ASCII 码（用十六进制数表示）：

(1) Example 1；　　(2) XiDian University；　　(3) －108.652；

(4) How are you?；　　(5) Computer；　　(6) Internet Web。

1.17　将下列字符串表示成相应的 ASCII 码（用十六进制数表示）：

(1) Hello；　　(2) 123＜CR＞456（注：＜CR＞表示回车）；

(3) ASCII；　　(4) The number is 2315。

第 2 章　8086CPU 结构与功能

微处理器（micro processor）也称中央处理单元（central processing unit，CPU），它是由超大规模集成电路构成的具有运算功能的逻辑部件。以 CPU 为核心可以构成计算机系统，我们先介绍 CPU 的外部结构、内部结构及功能结构，然后讨论微处理器包含的寄存器，它是编写汇编语言程序的基础，最后简要介绍微处理器系统中的存储器和 I/O 组织。

8088CPU 的结构与 8086CPU 类似，本章主要介绍 8086CPU 的结构与功能。

2.1　微处理器的外部结构

微处理器的外部结构如图 2.1 所示。8086CPU 片有40个管脚，微处理器通

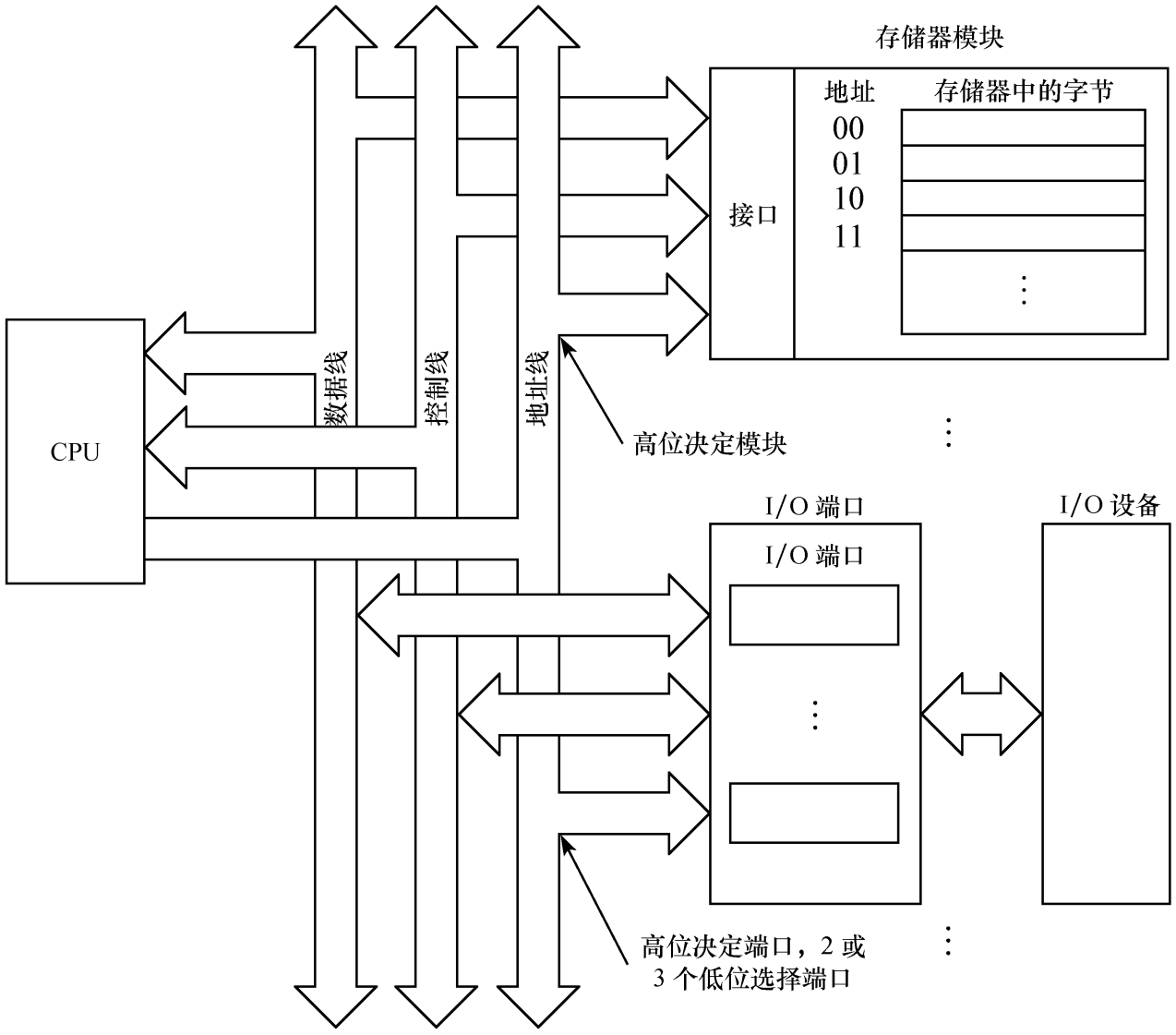


图 2.1　微处理器的外部结构

过这些引脚与外部的逻辑部件连接，完成信息的交换。CPU 的这些引脚信号称为微处理器级的总线，它应该能够完成下列功能：

（1）与存储器之间交换信息（指令及数据）。

（2）与 I/O 设备之间交换信息。

（3）能输入和输出必要的信号。

总线是用于连接 CPU 与其他部件的一组连线。总线从功能上可分为三种：

（1）数据总线（data bus）：传送信息。

（2）地址总线（address bus）：传送地址码。

（3）控制总线（control bus）：传送控制信号。

8086 CPU 有 16 条数据总线、20 条地址总线和 16 条控制总线。

存储器由几个模块组成，每个模块包含有多个存储单元，每个存储单元可存储指令和数据，每个存储单元都有一个唯一的地址，CPU 依据这个地址来存取指令和数据。用地址高位来区分模块。

I/O 接口是连接 CPU 与 I/O 设备的控制电路，在 I/O 接口中，有一个 I/O 端口寄存器，用于与 CPU 之间的数据交换，计算机也为其分配一个地址（端口地址），CPU 也是依据这个地址与端口打交道的。

存储器和 I/O 端口都是以字节为单位存放的。字符的 ASCII 码为 7 位代码，所以也用 8 位表示，一个字包含两个字节（16 位）。

某些微处理器采用统一的地址空间对存储器和 I/O 端口寻址，即存储器和 I/O 端口进行统一的地址编码，一个地址要么对应于存储单元，要么对应于端口寄存器，读写控制信号用来区分 CPU 是进行读/写操作。在这种方式下，对存储器和 I/O 端口的存取指令是一样的。

但大多数微处理器则是采用两个独立的地址空间，即存储器地址空间和 I/O 地址空间，这时，某存储单元和 I/O 端口可能对应于相同的地址值。那么如何区分 CPU 是存取存储单元还是 I/O 端口？采用存储器读写信号和 I/O 读写信号来区分。在这种方式下，对存储器和对 I/O 端口读写指令是不同的。

在 8086 微机系统中，采用 20 位地址对存储器进行编址，可寻址的地址范围为 $2^{20}=1M$；采用低 16 位地址对 I/O 端口编址，所以可寻址 $2^{16}=65536$ 个端口寄存器。

2.2　微处理器的内部结构

微处理器是组成计算机的核心部件，它具有下列运算和控制功能：

（1）进行算术和逻辑运算。

（2）具有接收存储器与 I/O 接口来的数据和发送数据给存储器与 I/O 接口的能力。

(3) 可以暂存少量数据。

(4) 能对指令进行寄存、译码并执行指令所规定的操作。

(5) 能提供整个系统所需的定时和控制信号。

(6) 可响应 I/O 设备发出的中断请求。

典型的 CPU 内部结构如图 2.2 所示。

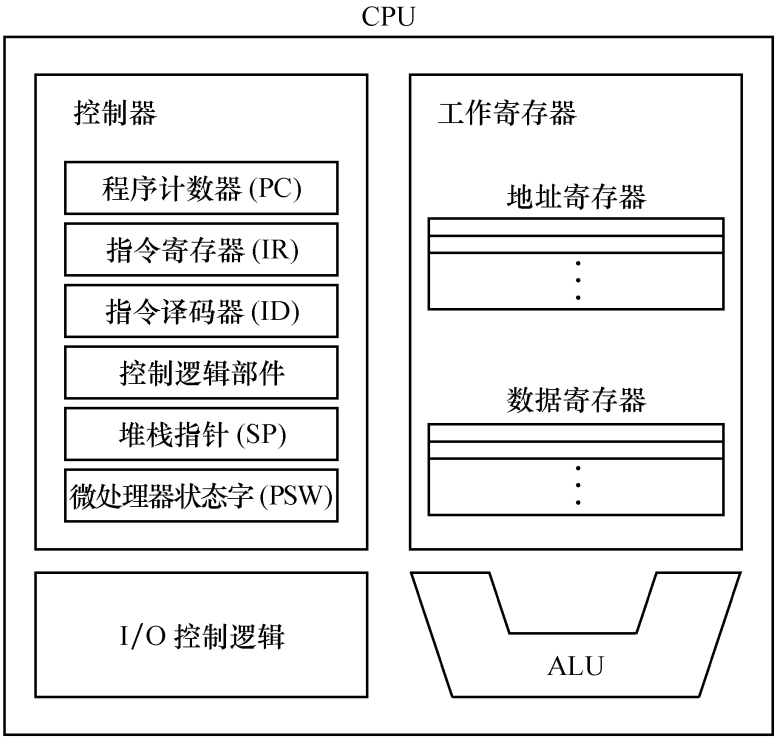


图 2.2 微处理器的内部结构

从 CPU 的内部结构可以看出，CPU 由四部分构成：算术逻辑运算单元 (ALU)、工作寄存器、控制器和 I/O 控制逻辑。

(1) 算术逻辑运算单元 ALU (arithmetic/logic unit)：它是运算器的核心，完成所有的运算操作。它是一个组合电路，无记忆功能。它有两个输入端和一个输出端，在控制信号的控制下可以完成不同的操作。

(2) 工作寄存器：可以暂存寻址信息和计算过程中的中间结果。数据寄存器用于暂存操作数及中间结果，地址寄存器用于暂存操作数的寻址信息。

(3) 控制器：它是 CPU 的“指挥中心”，完成指令的读入、寄存和译码，并产生控制信号序列，使 ALU 完成指定的操作。

控制器由下列部件组成：

① 程序计数器 (program counter，PC)：用于保存下一条要执行的指令的地址，也称指令指针 (instruction pointer)；

② 指令寄存器 (instruction register，IR)：保存从存储器中读入的当前要执行的指令；

③ 指令译码器 (instruction decoder，ID)：对指令进行译码；

④ 控制逻辑部件：根据对指令译码的分析，产生控制信号，以完成指令规定的操作；

⑤ 微处理器状态字（processor state word，PSW）：寄存处理器当前的状态，指令结果是否为 0，结果是正是负，有没有进位、借位，是否溢出等状态；

⑥ 堆栈指针（stack pointer，SP）：指示堆栈的地址。

(4) I/O 控制逻辑：处理 I/O 操作。

2.3 微处理器的功能结构

微处理器的功能结构如图 2.3 所示，它主要包含两个独立的逻辑单元：执行单元 EU（execution unit）和总线接口单元 BIU（bus interface unit）。ALU 的数据总线（16 位）、队列总线用于 EU 内部、EU 与 BIU 之间的通信。

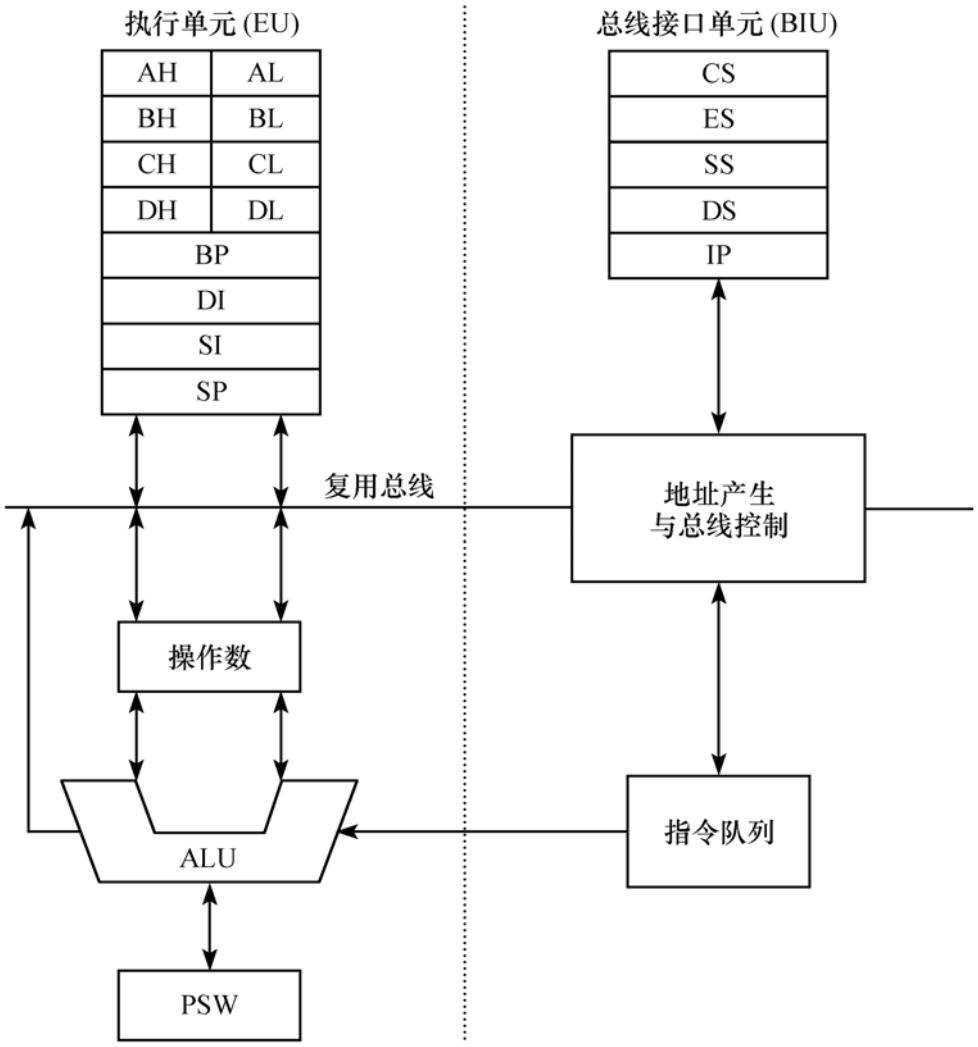


图 2.3 微处理器的功能结构

EU 的功能是执行指令规定的操作，主要部件有：算术逻辑运算单元（ALU）、暂存器、EU 控制器、微处理器状态字（PSW）和通用寄存器组。BIU 主要完成 CPU 与存储器和 I/O 设备之间的信息传递，主要部件有：算术逻辑运

算单元（ALU）、段寄存器、指令指针（IP，也称为 PC）、内部寄存器、指令队列寄存器和总线控制电路等。

BIU 主要完成取指令、存取数据的操作，其中 ALU 用于计算 20 位的指令或数据的地址，读取的指令代码存入指令队列寄存器，读取的数据通过 ALU 总线直接送给 EU。而 EU 直接从指令队列寄存器中获取指令，通过寄存、译码产生控制信号，完成指令规定的操作。

EU 和 BIU 可以独立、并行执行，但相互之间会有协作。当指令队列中还没有指令时，EU 处于等待状态，当 EU 执行指令需要访问存储器或 I/O 端口时，BIU 应尽快完成存取数据的操作，这一过程可以用图 2.4 表示。

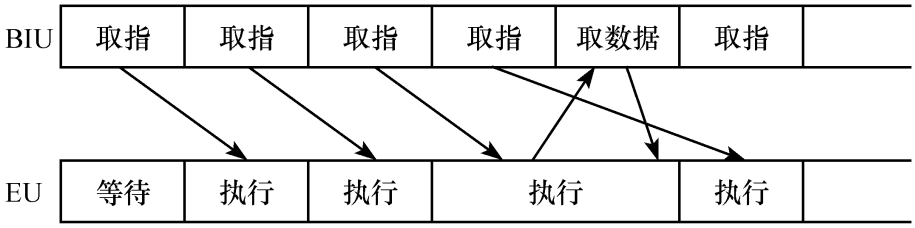


图 2.4　EU 和 BIU 单元的并行执行过程

EU 和 BIU 单元执行过程中，应该满足下列规则：

（1）当指令队列寄存器中无指令时，EU 处于等待状态。

（2）当指令队列中存满指令，而 EU 又没有访问存储器和 I/O 端口的需要，则 BIU 进入空闲状态。

（3）当指令队列中有两个空闲字节，则 BIU 自动执行取指令的总线周期。

（4）在 EU 执行指令时，需要访问存储器或 I/O 端口，如果这时 BIU 正在取指令，则应等待 BIU 完成取指令周期，然后 BIU 进入存储器和 I/O 端口访问周期。

（5）在 EU 执行转移、子程序调用或返回等指令时，自动清除指令队列的内容。

2.4　微处理器的寄存器组织

8086CPU 内部有 14 个 16 位的寄存器，按功能可以分成 8 个通用寄存器、4 个段寄存器和 2 个控制寄存器。寄存器是汇编语言指令可以使用的操作数，具有至关重要的地位，我们应该切实掌握。

2.4.1　通用寄存器

通用寄存器可以分成两类：数据寄存器（AX、BX、CX、DX）和地址指针/变址寄存器（SI、DI、SP、BP）。

8086CPU 有 4 个 16 位数据寄存器：

(1) 累加器 AX (accumulator)：这是最常用的寄存器，许多操作都可以在 AX 中完成，而且有一些操作只能在 AX 中完成，例如乘法和除法操作。

(2) 基址寄存器 BX (base register)：虽然属于数据寄存器，但它经常用作地址寄存器。

(3) 计数寄存器 CX (count register)：经常用作循环的计数寄存器，例如在循环语句中，默认 CX 的内容为循环次数。

(4) 数据寄存器 DX (data register)：用于寄存数据，但在 I/O 指令中，DX 用于表示端口地址。

另外，这 4 个 16 位的数据寄存器又可以分成 8 个 8 位的寄存器：

AX→AH,AL

BX→BH,BL

CX→CH,CL

DX→DH,DL

其中“H”表示高 8 位 (高字节)，“L”表示低 8 位 (低字节)，在程序设计中可以独立使用这 8 个字节寄存器。

8086CPU 有 4 个 16 位的地址指针/变址寄存器：

(1) 变址寄存器 SI (source index)：在字符串操作指令中，SI 提供源操作数的段内偏移地址，当然也可以在其他指令中，用作地址寄存器。

(2) 变址寄存器 DI (destination index)：在字符串操作指令中，DI 提供目的操作数的段内偏移地址，当然也可以在其他指令中，用作地址寄存器。

(3) 堆栈指针 SP (stack pointer)：用于保存堆栈段的段内偏移地址。段地址由段寄存器 SS 提供。堆栈是一块特殊的内存区域，它以“先进后出”的方式保存各寄存器、返回地址等信息。

(4) 基址指针 BP (base pointer)：BP 可以指定段内偏移地址，但将 BP 用作地址寄存器时，一般情况下，其默认的段地址为 SS。

在将这 4 个寄存器 (SI，DI，SP，BP) 用作地址寄存器时，它们只提供了 16 位的偏移地址。要形成 20 位的物理地址，还需要由段寄存器提供段地址，它们之间的关系为

$$\text{物理地址}=\text{段地址}\times 10\text{H}+\text{偏移地址}$$

2.4.2 段寄存器

8086CPU 的段寄存器有 4 个：

(1) 代码段寄存器 CS (code segment)：用于存放当前执行程序的段地址，IP 为指令指针。

(2) 数据段寄存器 DS (data segment)：用于存放当前数据段的段地址。

(3) 附加段寄存器 ES (extra segment)：用于存放当前附加数据段的段地址。

(4) 堆栈段寄存器 SS (stack segment)：用于存放当前堆栈段的段地址。

由段地址和偏移地址可以构成物理地址，这一点与存储器的分段结构有关，见第 2.5 节。

2.4.3 控制寄存器

8086CPU 有 2 个控制寄存器：

(1) 指令指针 IP (instruction pointer)：也称程序计数器 PC (program counter)，用于保存下一条即将要执行指令的段内偏移地址。改变 IP 的值就意味着改变程序的流程，不能通过普通的传送类指令修改 IP 的值，但某些指令可以改变 IP 的内容：比如转移指令、子程序调用和返回指令等。

(2) 微处理器状态字 PSW (processor state word)：它是一个 16 位的寄存器，一共设定了 9 个标志位，其中 6 个标志位用于反映 ALU 前一次操作的结果状态，另 3 个标志位用于控制 CPU 操作。具体位置如图 2.5 所示。

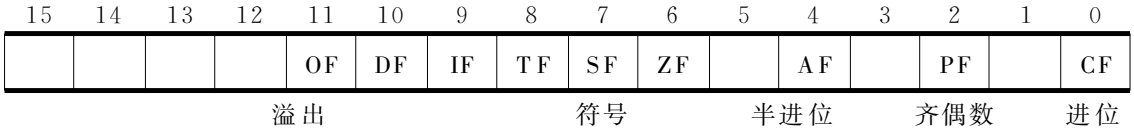


图 2.5 PSW 中的标志位

反映 ALU 前一次操作结果状态的标志位有：

① 进位标志 CF (carry flag)：在加减运算时，最高位 (D7 或 D15) 有无进(借) 位的标志。

$$\begin{cases} \text{有进(借)位时，} & CF = 1 \\ \text{无进(借)位时，} & CF = 0 \end{cases}$$

② 奇偶标志 PF (parity flag)：操作结果的低 8 位中含有 “1” 的个数。

$$\begin{cases} \text{偶数个“1”，} & PF = 1 \\ \text{奇数个“1”，} & PF = 0 \end{cases}$$

③ 辅助进位标志 AF (auxiliary carry flag)：在加减运算时，D3 位有无进(借) 位的标志。

$$\begin{cases} \text{有进(借)位时，} & AF = 1 \\ \text{无进(借)位时，} & AF = 0 \end{cases}$$

④ 零标志 ZF (zero flag)：运算结果是否为 0。

$$\begin{cases} \text{结果为 0，} & ZF = 1 \\ \text{结果不为 0，} & ZF = 0 \end{cases}$$

⑤ 符号标志 SF (sign flag)：操作结果的符号，它等同于操作的最高位 D7

(或 D15)。

$$\begin{cases} \text{操作结果为负，} & SF = 1 \\ \text{操作结果为正，} & SF = 0 \end{cases}$$

⑥ 溢出标志 OF（overflow flag)：有符号数运算时是否溢出的标志。

$$\begin{cases} \text{溢出，} & OF = 1 \\ \text{无溢出，} & OF = 0 \end{cases}$$

例如，将下列两个二进制数相加，可以设定各个标志位

$$\begin{array}{rcccc} & 0101 & 0100 & 0011 & 1001B \\ +) & 0100 & 0111 & 0110 & 1010B \\ \hline & 1001 & 1011 & 1010 & 0011B \end{array}$$

结果有：SF=1，ZF=0，PF=1，AF=1，CF=0，OF=1。

又如，将下列两个二进制数相减，并设定标志位

$$\begin{array}{rcccc} & 0101 & 0110 & 0101 & 1011B \\ -) & 1100 & 0101 & 0100 & 0110B \\ \hline [1] & 1001 & 0001 & 0001 & 1101B \end{array}$$

结果有：SF=1，ZF=0，PF=1，AF=0，CF=1，OF=1 。

控制 CPU 的标志位有：

① 方向标志 DF（direction flag)：在字符串操作中，当 DF=0 时，其变址寄存器（SI，DI）的内容自动递增；当 DF=1 时，SI、DI 自动递减。

② 中断允许标志 IF（interrupt enable flag)：当 IF=1 时，CPU 能够响应可屏蔽中断请求；当 IF=0 时，则 CPU 不能响应中断请求，这一位可以用指令（STI、CLI）来设置。

③ 陷阱标志 TF（trap flag)：当 TF=1 时，则 CPU 处于单步执行方式，即每执行一条指令就自动执行一次类型 1 的内部中断，这主要用在 Debug 中。

2.5 微处理器的存储器和 I/O 组织

2.5.1 存储器地址空间和数据存储格式

在 8086CPU 构成的系统中，存储器单元的大小为一个字节，每个存储单元都对应于唯一的一个地址。由于微处理器系统有 20 条地址线 $A_0 \sim A_{19}$，可寻址范围达 2^{20}（1M)，因此一共可以设计 1M 个字节的存储空间。相邻的两个字节构成一个字，低地址存储低字节，高地址存储高字节。

当一个字从偶地址开始存储时，则称为字的存储是对准的；否则，当一个字从奇地址开始存储，则称为字的存储是未对准的。这一点与 CPU 访问一个字时

的总线周期（通过总线访问一次存储器或 I/O 端口的时间）有密切关系，从理论上说，由于 8086CPU 具有 16 条数据总线，CPU 的一个总线周期可以存取一个字，但实际上，只有当字的存储是对准时，CPU 存取一个字仅需要一个总线周期；当字的存储是未对准时，CPU 存取这个字时需要两个总线周期（第一个总线周期先访问低字节，第二个总线周期访问高字节，当然这个过程是 CPU 自动完成的，对用户来说，字的存储方式与编程没有任何关系）。

2.5.2 存储器的分段和物理地址的形成

前面已经提到过，物理地址由段地址和段内偏移地址两部分组成。由于基址或变址寄存器为 16 位的寄存器，它们可以提供 16 位的偏移地址，因此通过改变基址或变址寄存器可以寻址 2^{16} =64KB 的存储空间。8086CPU 采用地址分段的方法，使寻址范围扩大到 1M 字节。

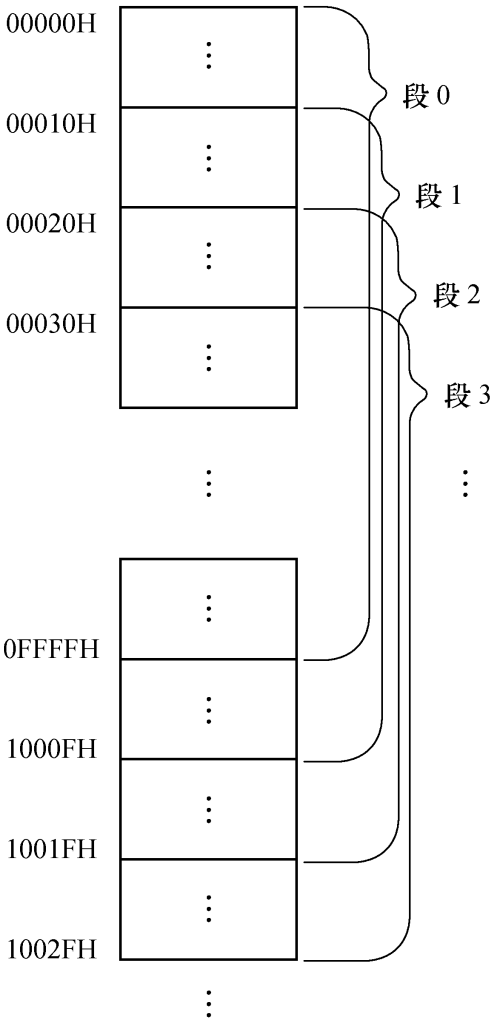


图 2.6 存储器空间的分段组织

在 8086CPU 系统中，把 1MB 的存储器空间划分成若干个逻辑段，每个段最多有 64KB 的存储空间，各段起点选在能够被 16 整除的地址，并将高 16 位的地址值称为段地址，因此 1M 字节的存储器空间可以有 2^{16} =64K 个段，但段与段之间是相互覆盖的，相邻段之间相距 16 个存储单元，如图 2.6 所示。

由于段与段之间是相互覆盖的，因此同一个地址的存储单元可表示成不同的段地址和偏移地址，例如物理地址为 00054H 的存储单元，可以表示成段 0＋0054H、段 1＋0044H、段 2＋0034H、段 3＋0024H、段 4＋0014H、段 5＋0004H，只要满足段地址×10H＋偏移地址能够给出正确的物理地址。存储单元的物理地址的计算过程如图 2.7 所示，这里段地址×10H 等效于向左移 4 位，BIU 中的 ALU 就是专门用来完成物理地址的计算的。

物理地址可以表示成逻辑地址格式：（段地址：偏移地址），例如逻辑地址 0800H：0100H 相对应的物理地址为 08100H，逻辑地址 5421H：256AH 的物理地址为 5677AH。

CPU 可通过四个段寄存器（CS，DS，SS，ES），访问 4 个不同的段，这些段称为当前段，它们可用来分类存储信息。在汇编语言程序设计中，存储器中的信息可分成三类：程序（指令）代码、数据和状态信息，可以为它们分别分配一