汇编语言程序设计

張青林　主编

上海科学技术出版社

前言 PREFACE

當今時代，計算機技術已成爲每個人必須掌握的文化基礎。計算機科學家們（或專家們）爲了使用的方便和普及的需要，爲廣大使用者構建了由“窗口＋圖標＋按鈕＋自然語言表述”的界面。它給人們帶來方便的同時，也讓“界面”遮擋了人們的視野。如果要進一步用好、用活計算機，開掘它潛在的軟、硬件資源，就必須透過這一“界面”，向更深層次進發，提升計算機知識的深度和廣度。“匯編語言程序設計”就是能拔高計算機知識層次，擴大專業相關知識面的一門重要課程。它是大專院校計算機類、電子類、電氣類、自動化類等專業必修的核心課程之一，是“微機原理及應用”、“微機接口技術”、“操作系統”等其他主干專業課的先修課。本課程對于訓練學生理解信息處理過程和原理，掌握程序設計技術，熟悉上機及程序調試技術都有着重要作用。而且，該課程還是國家計算機等級考試 PC 技術課程之一。

本書以 Intel 8086/8088 爲基本機型，詳細介紹了 8086/8088 微處理器的尋址方式和指令系統，結合程序設計的需要，介紹了常用僞指令；以順序程序、分支程序、循環程序和子程序設計爲引領，詳細講解了編制匯編語言程序的要領、方法和技巧。在本書的最后部分，介紹了 80x86/Pentium 部分新增指令和 80x86 的編程技術。爲適應不同基礎學生的需求，兼顧提高學習興趣的考慮，增加了匯編語言程序設計的綜合舉例和練習，供學有余力的學生選學。

在長期的教學實踐中，我們深深感到，由于“匯編語言程序設計”這門課程自身的特點，學生學習有難度，普遍覺得理解記憶的分量較重而且對理論教學與上機操作的銜接不滿意。如果編制源程序后，用匯編程序（或稱匯編器）匯編，從教學進程上講，必須講完指令和僞指令之后，才能介紹匯編語言程序上機過程，從而導致理論教學與實踐教學脫節，即上機操作在時間上后延較長，一般要到后半學期才能夠上機，這無論是對教還是對學都是不利的。針對這一點，我們在教材的內容遴選，實踐教學安排上作了有益的嘗試和改革，充分體現以人爲本，尊重學生對教學的合理要求，力求達到以下特點：

1. 選擇合適的教學基點，取材適宜，內容精練，突出重點，化解難點。比如，將“指令系統”化整爲零：即先介紹常用基本指令，形成基本的知識脈絡。

而把轉移指令、循環指令及子程序調用與返回指令，插入到相關程序設計章節中去講解，旨在消除學生連續學習指令系統，造成知識容量大，純理論教學的那種單調乏味的感覺。

2. 上機操作內容分爲三個方面：

（1）在啓動 DEBUG 之后，做寄存器、存儲器操作。

（2）在 DEBUG 環境下做基本指令的操作、簡單程序的運行、簡單問題的程序編制與調試。

（3）利用匯編器進行程序的編制、程序的上機調試運行。這是我們在已有的實踐教學基礎上作出的安排，雖然 DEBUG 下操作不是我們教學的立足點，但能讓學生盡早上機操作，及時驗證、運用和鞏固所學知識。實踐證明，這樣做與教學的銜接很緊密，教學效果明顯改善。

3. 大量的程序設計舉例基本上都配有程序流程圖，旨在將理論叙述與形象直觀圖示相結合，便于學生理解接受。

書中打“＊”的章節爲選學內容。

在教學具體安排上，編者建議用 48 學時講授基本內容（第 1～第 8 章），講完第 2 章即能安排學生上機操作，上機可安排 24 學時。如有更多學時，可選講第 9、第 10 章，或安排學生自學。

本書可作爲高等專科學校、職業技術學院、成人高等院校、本科院校舉辦的職業技術學院計算機、網絡技術、信息管理、應用電子、電氣技術、測控技術與儀器等專業的教材，也可供其他專業學生和有關專業技術人員參考，或作爲自學用書。

本書由安徽經濟管理學院張青林任主編，李明才、何強、楊振宇任副主編，安徽水利水電職業學院汪永華任主審。書中第 1、第 2 章和附録 1 和 2 由安徽水利水電職業學院何強編寫；第 3、第 9 章由安徽職業技術學院李明才編寫；第 4、第 10 章和上機實驗指導由安徽交通職業技術學院楊振宇編寫；第 5、第 6、第 7、第 8 章及上機實驗指導的部分內容由張青林編寫，并由張青林對全書進行了統稿。汪永華對全書作了仔細的審閱和修改，在此深表感謝！

由于編者水平所限，書中錯誤、不妥之處在所難免，懇請各位專家和讀者提出寶貴意見。

編　者

目录 CONTENTS

第 1 章
汇编语言及相关基础知识

>> 本章学习要点：

(1) 掌握有關匯編語言的基本概念，包括：程序、指令的概念，匯編過程及匯編語言的特點、應用場合等。

(2) 掌握常用數制及其轉換，包括：二進制、十進制、八進制、十六進制之間的轉換，以及 8421BCD 碼和 ASCII 碼。

(3) 理解計算機中數的表示方法，以及有關帶符號數、無符號數、原碼、補碼、反碼和運算中的溢出問題。

(4) 掌握基本的邏輯運算，包括：與運算、或運算、非運算和异或運算。

1.1 汇编语言概述

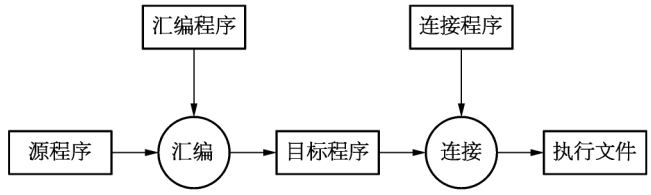
1.1.1 汇编语言的概念

汇编语言是一种面向机器的语言，其指令与机器指令是一一对应的。汇编语言用符号和文字来表示指令，所以它又称为符号语言。

所谓指令是指控制计算机执行某一特定操作的命令。一台计算机所能识别的指令的全体称为指令系统，它反映了计算机基本功能的强弱。

而机器指令是以二进制代码形式表示的、能直接为计算机识别并执行的命令，它通常由操作码和操作数两部分组成。8086/8088 的指令系统有 100 多条基本指令。

程序是按照确定的算法解决具体问题所必需的指令序列，由数据、指令和字符等构成，在程序执行前应预先将其以二进制代码的形式存储在存储单元中。用汇编语言编写的程序是不能被计算机直接识别和执行的(同用高级语言编写的程序一样)，它需要翻译成目标程序后方可执行，这个过程我们称为汇编(如“由汇编语言源程序到执行文件的处理过程图”所示)。汇编语言的使用虽然不如高级语言简单方便，但因它与机器语言是一一对应的，故可充分利用计算机硬件系统的特性，提高编程技巧和编程质量。另外，利用汇编语言处理 I/O 设备是汇编语言的独到之处，所以它是无法被其他语言所取代的。汇编程序的类型有：自汇编程序、交叉汇编程序、微汇编程序、浮动汇编程序和宏汇编程序。



由汇编语言源程序到执行文件的处理过程图

汇编语言(ASM)虽然较机器语言在阅读、记忆及编写方面都提高了很多,但对描述任务、编程设计而言仍不方便,于是产生了具有机器语言优点,又能较好地面向问题的语言,即宏汇编语言(MASM)。

宏汇编语言不仅包含一般汇编语言的功能,而且用了高级语言使用的数据结构,是一种接近高级语言的汇编语言。例如,它提供了记录、结构和字符串操作;具有宏处理、条件汇编及磁盘操作系统 DOS 功能调用等多种功能;程序的开发以及调试手段也比较完善,因此,宏汇编语言是一种更高级的汇编语言。

1.1.2 汇编语言的特点

汇编语言相对机器语言而言易记好用,但远不如高级语言方便、实用,而且编写同样的程序,使用汇编语言比使用高级语言花费的时间更多,调试和维护更困难。既然如此,为什么还要使用汇编语言呢?主要有两个原因,即语言的性能和对计算机的完全控制能力。汇编语言具有如下五个特点:

1. 執行速度快

汇编语言程序比高级语言程序执行得更快,而程序的执行速度对于某些应用来说是至关重要的。对于这些应用,单纯使用高级语言往往达不到速度要求,单纯使用汇编语言编写程序也不是最好的方案,所以,许多成功的大型应用程序往往使用混合编程。首先使用高级语言编写整个程序,然后测试程序的执行时间,再使用汇编语言重写其中最费时间的部分。这样做的依据是,在实际使用中,程序的大部分执行时间通常都花费在一小部分代码上。

2. 程序體積小

汇编语言程序的体积比高级语言程序更小。在某些情况下,设备中的嵌入式处理器往往只有很少的内存,例如,智能卡中有 CPU,但是智能卡中很难有 1 MB 以上的内存,也不可能有带分页的硬盘,而智能卡又必须执行复杂的加密解密计算,使用汇编语言可能是惟一的方法。个人数字助理(PDA)和其他使用电池作为能源的无线电子设备,为了节省电池的电力,往往也只有很少的内存,它们也需要使用短小精悍、高效率的机器代码。

3. 可以直接控制硬件

某些应用程序要求能够完全控制计算机硬件,这也必须使用汇编语言。如操作系统中的低级中断和陷阱处理程序,以及许多嵌入式实时系统中的设备控制程序等,都属于这一类应用。

4. 可以方便地編譯

编译器可以产生供编程者使用的汇编程序，或者自己执行汇编过程。因此，为了理解编译器的工作原理，必须首先理解汇编语言。

5. 輔助計算機工作者掌握計算機體系結構

研究汇编语言可以使人们掌握计算机的实际结构，特别是对于学习计算机体系结构的人员，编写汇编语言是在结构层理解计算机的惟一途径。

1.1.3 汇编语言的应用场合

随着软件技术的发展，汇编语言的应用领域越来越小，但是它的特点又决定了它的必然作用。下面罗列了汇编语言的部分应用场合：

（1）程序要具有较快的执行时间，或者只能占用较小的存储容量。例如，操作系统的核心程序段，实时控制系统的软件，智能仪器仪表的控制程序等。

（2）程序与计算机硬件密切相关，程序要直接、有效地控制硬件。例如，I/O 接口电路的初始化程序段，外部设备的低层驱动程序等。

（3）大型软件需要提高性能、优化处理的部分。例如，计算机系统频繁调用的子程序和动态链接库等。

（4）没有合适的高级语言或只能采用汇编语言的时候。例如，开发最新的处理器程序时，暂时没有支持新指令的编译程序等。

汇编语言还有许多其他的实际应用，例如，分析具体系统尤其是该系统的低层软件、加密解密软件、分析和防治计算机病毒等。

1.2 常用数制及其转换

1.2.1 常用数制

数制是数的表示及计算方法。数制型数据是有大小的，人们习惯使用的是十进制数，但在计算机内，各种信息都是用二进制代码形式表示的。为了书写方便，二进制数也常用八进制数和十六进制数表示。但任何一种数制都具有以下几个要点：

（1）基数：每种进制的基数就是该进制本身，如二进制的基数是 2。

（2）数码：二进制的数码为 0，1；十进制的数码为 0～9；八进制的数码为 0～7；十六进制的数码为 0～9、A～F 等。

（3）进位原则：十进制逢十进一，二进制逢二进一等。

（4）位权：指每一位数位上数码所具有的权，如十进制的位权为 10^i，二进制的位权为 2^i 等，i 的取值为整数。

四种常用进制数的关系如表 1－1 所示。

表 1－1　四种常用进制数的对照表

十进制	二进制	八进制	十六进制	十进制	二进制	八进制	十六进制
0	0000	0	0	8	1000	10	8
1	0001	1	1	9	1001	11	9
2	0010	2	2	10	1010	12	A
3	0011	3	3	11	1011	13	B
4	0100	4	4	12	1100	14	C
5	0101	5	5	13	1101	15	D
6	0110	6	6	14	1110	16	E
7	0111	7	7	15	1111	17	F

由于不同位置的权值不同，因此同一数码在不同位置上，表示的值也是不同的。每个数位上的值等于该位置上的数码与该位置的权值的乘积，相邻数位中，高位权与低位权之比即是该进制的基数。利用下式，可将任意进制的数转化为十进制数，其中，a 表示某数制的数码，R 表示基数(底)，R^i 表示数位的权，m 和 n 为正整数。

$$N=\sum_{i=-m}^{n=1}a_iR^i=a_{n-1}R^{n-1}+a_{n-2}R^{n-2}+\cdots+a_1R^1+a_0R^0+\cdots+a_{-m}R^{-m}$$

1.2.2　数制的转换

下面具体介绍各种数制之间的转换方法。

1. 将十进制数轉換成二進制数、八進制数或十六進制数

转换方法：整数部分除以基数(2、8 或 16)取余数，小数部分乘以基数(2、8 或 16)取整数。

【例 1.1】　试将十进制数 23.125 转换成二进制数。

解：(1) 整数部分的转换：除 2 取余。

根据“除 2 取余”法的原理，按如下步骤转换：

2 | 23 ……余 1　b_0
2 | 11 ……余 1　b_1
2 | 5 ……余 1　b_2
2 | 2 ……余 0　b_3
2 | 1 ……余 1　b_4
0

读取顺序

则：$(23)_{10}=(10111)_2$

(2) 小数部分的转换：乘2取整。

小数部分的转换正好与整数部分的转换相反，先将0.125乘以2得结果0.25，则在右边记下整数位0，再将去整数位后的0.25乘以2，记下结果的整数位，依此类推，直到最后小数部分为0或结果已达到精度要求为止。最后把刚记下的整数位顺着排列即得结果，用“乘2取整”法，按如下步骤转换：

$0.125\times2=0.25$ ……0 b_{-1}

$0.25\times2=0.5$ ……0 b_{-2}

$0.5\times2=1$ ……1 b_{-3}

因此：$(0.125)_{10}=(0.001)_2$

所以，十进制数23.125转换成二进制数为10111.001。十进制数转换成八进制数或十六进制数的方法同上，$(45.25)_{10}=(55.2)_8=(37.4)_{16}$。

2. 將二進制數、八進制數或十六進制數轉換成十進制數

转换方法：按位权展开求和。

【例1.2】 $(10111)_2=1\times2^4+0\times2^3+1\times2^2+1\times2^1+1\times2^0=(23)_{10}$

$(572.34)_8=5\times8^2+7\times8^1+2\times8^0+3\times8^{-1}+4\times8^{-2}=(378.4375)_{10}$

$(35A)_{16}=3\times16^2+5\times16^1+10\times16^0=(858)_{10}$

3. 二進制數與八進制數、十六進制數的轉換

因为$2^3=8$，所以二进制数转换成八进制数只需将二进制数从小数点开始每3位转换成1位八进制数(整数由左向右，小数相反，不足3位的用0补齐)。

【例1.3】 $(101\ 111\ 010.\ 011\ 100)_2=(572.34)_8$

八进制数转换成二进制数只需将每1位八进制数用3位二进制数表示，小数点位置不变。

【例1.4】 $(175.54)_8=(001\ 111\ 101.101\ 100)_2=(1111101.1011)_2$

十六进制数与二进制数的转换只需将上面二进制数与八进制数转换方法中每隔3位改为每隔4位即可。

【例1.5】 $(9B28)_{16}=(1001\ 1011\ 0010\ 1000)_2=(1001101100101000)_2$

1.3 无符号数和有符号数

各种数据在计算机中的表示形式称为机器数，其特点是数的符号用“0”和“1”表示，其中，“0”表示正号，“1”表示负号，小数点隐含表示而不占位置。机器数对应的实际数值称为该机器数的真值。

机器数分为无符号数和有符号数两种。无符号数表示正数，在机器数中没有符号位。对于无符号数，若约定小数点的位置在机器数的最低位之后，则是纯整数；若约定小数点的位置在机器数的最高位之前，则是纯小数。对于有符号数，机器数的最高位是表示正、负的符号位，其余二进制位表示数值。若约定小数点的位置在机器数的最低位之后，则是纯整

数；若约定小数点的位置在机器数的最高位之前（符号位之后），则是纯小数。这种表示数的方式称为定点数。

定点数表示法的缺点在于其形式过于僵硬，固定的小数点位置决定了固定位数的整数部分和小数部分，不利于同时表示特别大的数或者特别小的数，所以绝大多数现代的计算机系统采用了所谓的浮点数表示方式。这种表示方式利用科学计数法来表示实数，即用一个尾数（M）、一个基数（B）、一个指数（E）和一个表示正负的符号来表示实数，直观表示为 $N=M\times B^E$。如 123.45 用十进制科学计数法可以表示为 1.2345×10^2，其中 1.2345 为有效数字（即尾数），10 为基数，2 为指数。浮点数利用指数达到了浮动小数点的效果，从而可以灵活地表示更大范围的实数。

1.4 原码、反码和补码

为了运算方便，机器数有不同的编码方法，称为码制。常用的码制有原码、反码、补码和移码等。

1. 原碼

原码又称为符号绝对值码。数据最高位为符号位，正数用“0”表示，负数用“1”表示。其他位为数据位，用二进制数的绝对值表示。原码与真值转换方便，但做加减运算不方便，而且零有“+0”和“−0”两种表示方法。

【例 1.6】 当机器字长为 8 位时：

$[+1]_{原}=00000001B$　　$[-1]_{原}=10000001B$

$[+127]_{原}=01111111B$　　$[-127]_{原}=11111111B$

$[+0]_{原}=00000000B$　　$[-0]_{原}=10000000B$

2. 反碼

正数的反码表示与原码相同；负数的反码，符号位为“1”不变，数值位由其绝对值各位取反得到。零的反码也有“+0”和“−0”两种表示方法。反码因运算不便而较少使用。

【例 1.7】 当机器字长为 8 位时：

$[+1]_{反}=00000001B$　　$[-1]_{反}=11111110B$

$[+127]_{反}=01111111B$　　$[-127]_{反}=10000000B$

$[+0]_{反}=00000000B$　　$[-0]_{反}=11111111B$

3. 補碼

为了加减运算的方便引入了补码概念，其关键思想是用加法代替减法。正数的补码与原码表示相同；负数的补码，符号位用“1”表示，数值位用其绝对值的补码表示（即原码各位求反，末位加 1）。在补码表示中，0 有唯一的编码：$[+0]_{补}=[-0]_{补}=00000000B$。

【例 1.8】 当机器字长为 8 位时：

$[+1]_{补}=00000001B$　　$[-1]_{补}=11111111B$

$[+127]_{补}=01111111B$　　$[-127]_{补}=10000001B$

补码的表示范围比原码和反码略宽，在定点小数中，补码可以表示 −1，即 $[-1]_{补}=$

10000B，而原码和反码不能表示绝对值为1的数。

【例1.9】　假定机器字长为8位，试写出122的原码、反码和补码。

$[122]_{原}=[122]_{反}=[122]_{补}=01111010B$

【例1.10】　假定机器字长为8位，试写出−45的原码、反码和补码。

$[-45]_{原}=10101101B$

$[-45]_{反}=11010010B$

$[-45]_{补}=11010011B$

对于用补码表示的负数，首先认定它是负数，然后用求它的补码的方法得到它的绝对值，即可求得该负数的值。例如，补码数11110011B是一个负数，求该数的补码为00001101B，该数相应的十进制数为13，故求出11110011B为−13D。

4. 溢出

各种数据编码都有其数据表示范围，如果在运算过程中出现的数据超出编码表示的范围，称为溢出。对于加法，只有在正数加正数和负数加负数两种情况下才会产生溢出，即符号相同的两个数相加可能会产生溢出，而符号不同的两个数相加不会产生溢出。对于减法，只有在正数减负数和负数减正数两种情况下才会产生溢出，即符号不同的两个数相减可能会产生溢出，而符号相同的两个数相减不会产生溢出。

在微处理器中，使用补码进行运算是十分方便的，它使同一个微处理器中既能运算带符号数又能运算不带符号的数。而在进行带符号数的加减运算时，应把参与运算的数据转换成补码形式进行运算。当使用8位二进制数表示带符号的数时，它所能表示的数值范围在$(-128)_{10}\sim(+127)_{10}$之间，如果相加结果超出了这个范围，就会导致错误发生。

$[X+Y]_{补}=[X]_{补}+[Y]_{补}$

$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$

【例1.11】　两个带符号的数01000001B与01000011B相加：

$$\begin{array}{rr} & 01000001 \\ + & 01000011 \\ \hline & 10000100 \end{array}$$

本例中是两个正数相加，但结果却是一个负数(符号位为1)。显然，这个结果是错误的，出现这种错误的原因就在于这两个数相加的结果超过了8位二进制带符号数所能表示的数值范围。

【例1.12】　两个带符号的负数10001000B和11101110B相加：

$$\begin{array}{rr} & 10001000 \\ + & 11101110 \\ \hline \boxed{1} & 01110110 \end{array}$$

由于规定用8位二进制数来表示带符号的数，故忽略作为进位位的第九位。按8位二进制数来解释这两个带符号的负数的相加，其结果为一个正数。很明显，结果是错误的。

【例1.13】　两个无符号数11111101B和00000011B相加：

$$\begin{array}{rr} & 11111101 \\ + & 00000011 \\ \hline \boxed{1} & 00000000 \end{array}$$

从相加计算的结果来看，如果微处理器只有 8 位，也就是用 8 位二进制数来解释运算的结果，则将出现错误。因此，在微处理器中设有专门的一位，称为进位位，它将用于保存第九位，以防丢失信息。

1.5 BCD 码和 ASCII 码

1.5.1 BCD 码

计算机内部使用的是二进制数，但二进制数书写冗长，阅读不便，所以，在输入输出时人们仍习惯使用十进制数。如果计算量不大，可采用二进制数对每一位十进制数字进行编码的方法来表示一个十进制数，这种数叫做 BCD(Binary-Coded Decimal)码。由于在机器内采用 BCD 码进行运算，绕过了二进制、十进制间的复杂转换环节，从而节省了机器时间。

BCD 码有多种形式，最常用的是 8421BCD 码，它是用 4 位二进制数对十进制数的每一位进行编码，这 4 位二进制码的值就是被编码的一位十进制数的值。由于 4 位二进制数可以表示 16 种状态，所以丢弃最后 6 种状态，选用 0000～1001 来表示 0～9 十个数值。这种编码又叫做 8421 码，如表 1－2 所示。

表 1－2　十进制数与 8421BCD 码的对应关系

十进制数	BCD 码	十进制数	BCD 码
0	0000	10	00010000
1	0001	11	00010001
2	0010	12	00010010
3	0011	13	00010011
4	0100	14	00010100
5	0101	15	00010101
6	0110	16	00010110
7	0111	17	00010111
8	1000	18	00011000
9	1001	19	00011001

【例 1.14】　将十进制数 69.25 转换成 BCD 码。

6 9 . 2 5

0110 1001. 0010 0101

结果为 69.25＝(01101001.00100101)BCD

【例 1.15**】 将** BCD **码** 100101111000.01010110 **转换成十进制数。**

1001 0111 1000.0101 0110

9 7 8. 5 6

结果为(100101111000.01010110)BCD＝978.56

1.5.2 字符编码 (ASCII 码)

鉴于信息交换的重要性以及统一文字符号的编码标准,让不同厂家不同机型的计算机都能使用同一套标准化的信息交换码,于是美国国家标准局特别制定了 ASCII **码**(America Standard Code for Information Interchange,**美国信息交换标准码**),**作为数据传输的标准码。**

ASCII **码的每个字符用** 7 **位二进制数表示,其排列次序为** $d_6d_5d_4d_3d_2d_1d_0$, d_6 **为高位,** d_0 **为低位。而一个字符在计算机内实际是用** 8 **位表示。正常情况下,最高一位** d_7 **为** 0。7 **位二进制数共有** 128 **种编码组合,可表示** 128 **个字符,其中数字** 10 **个、大小写英文字母共** 52 **个、其他字符** 32 **个和控制字符** 34 **个。**

(1) **数字** 0～9 **的** ASCII **码为** 30H～39H。

(2) **大写英文字母** A～Z **的** ASCII **码为** 41H～5AH。

(3) **小写英文字母** a～z **的** ASCII **码为** 61H～7AH。

对于 ASCII **码表中的** 0、A、a **的** ASCII **码** 30H、41H、61H **应尽量记住,其余的数字和字母的** ASCII **码可按数字和字母的顺序以十六进制的规律写出,具体的** ASCII **码见表** 1－3。

表 1－3 ASCII 码表

$d_3d_2d_1d_0$ \ $d_6d_5d_4$		0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	、	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&.	6	F	V	f	v

（续表）

$d_3\,d_2\,d_1\,d_0$ \ $d_6\,d_5\,d_4$		0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	—	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	_	o	DEL

注：行为高三位，列为低四位，如“@”的 ASCII 码值为 0100 0000。

表 1－3 中控制字符的含义如表 1－4 所示。

表 1－4　控制字符含义

NUL	**空**	SOH	**标题开始**	STX	**正文开始**	ETX	**本文结束**	EOT	**传输结果**
ENQ	**询问**	ACK	**确认**	BEL	**振铃**	BS	**退一格**	HT	**横向列表**
LF	**换行符**	VT	**垂直制表**	FF	**换页**	CR	**回车**	SO	**移位输出**
SI	**移位输入**	DLE	**数据链换码**	DC1	**设备控制 1**	DC2	**设备控制 2**	DC3	**设备控制 3**
DC4	**设备控制 4**	NAK	**否定**	SYN	**空转同步**	ETB	**信息组传送结束**	CAN	**作废**
EM	**纸尽**	SUB	**减**	ESC	**换码**	FS	**文字分隔符**	GS	**组分隔符**
RS	**记录分隔符**	US	**单元分隔符**	DEL	**删除**	SP	**空格**		

1.6　基本逻辑运算

8086 中除了进行加、减、乘、除等基本算术运算外，还可对两个或一个无符号的二进制数进行逻辑运算，如进行两个数的比较，或者从某个数中选取某几位等操作。

8086 的逻辑运算主要包括：逻辑非、逻辑与、逻辑或和逻辑异或四种基本运算。

1. 邏輯非運算

逻辑非（NOT）又称“求反”，运算法则是按位进行求反，即“1”变为“0”，“0”变为“1”。常

在数的上方加一横线来表示，即 X 的逻辑非记作$\overline{X}$。

【例 1.16】 X=01011，$\overline{X}$=10100；Y=01001011，$\overline{Y}$=10110100

逻辑非可用“非”门电路实现。

2. 邏輯與運算

逻辑与(AND)又称“逻辑乘”，常用记号“∧”或“·”来表示。运算法则也是按位进行运算，两个对应位同时为 1 时，运算结果的对应位为 1，否则为 0。

【例 1.17】 1101∧1011=1001

逻辑与可用“与”门电路实现。

3. 邏輯或運算

逻辑或(OR)又称“逻辑加”，常用记号“∨”或“+”表示。运算法则也是按位进行。两个对应位同时为 0，运算结果的对应位为 0，否则为 1。

【例 1.18】 1101∨1001=1101

逻辑或运算可以由“或”门电路来实现。

4. 邏輯异或運算

逻辑异或(XOR)常用记号“⊕”表示。运算法则也是按位进行。两个对应位相同时，运算结果的对应位为 0，否则为 1。

【例 1.19】 1101⊕1011=0110

逻辑异或可用“异或”门电路实现。

习　　题

1. 簡述計算機系統的硬件組成及各部分作用。
2. 什么是匯編語言源程序、匯編程序和目標程序？
3. 匯編語言與高級語言相比有什么優缺點？
4. 完成下列十六進制數的減法：

(1) FFFF－AAAA=

(2) 12DF－02DA=

5. 將下列十六進制數分別轉換爲二進制數和十進制數表示：

(1) FFH　(2) 0H　(3) 5EH　(4) EFH

(5) 2EH　(6) 10H　(7) 1FH　(8) ABH

6. 將下列十進制數轉換爲 BCD 碼表示：

(1) 12　(2) 24　(3) 68　(4) 127

(5) 128　(6) 255　(7) 1234　(8) 2458

7. 將下列 BCD 碼轉換爲十進制數表示：

(1) 10010001　(2) 10001001　(3) 00110110　(4) 10010000

(5) 00001000　(6) 10010111　(7) 10000001　(8) 00000010

8. 將下列十進制數分別用 8 位二進制數的原碼、反碼和補碼表示：