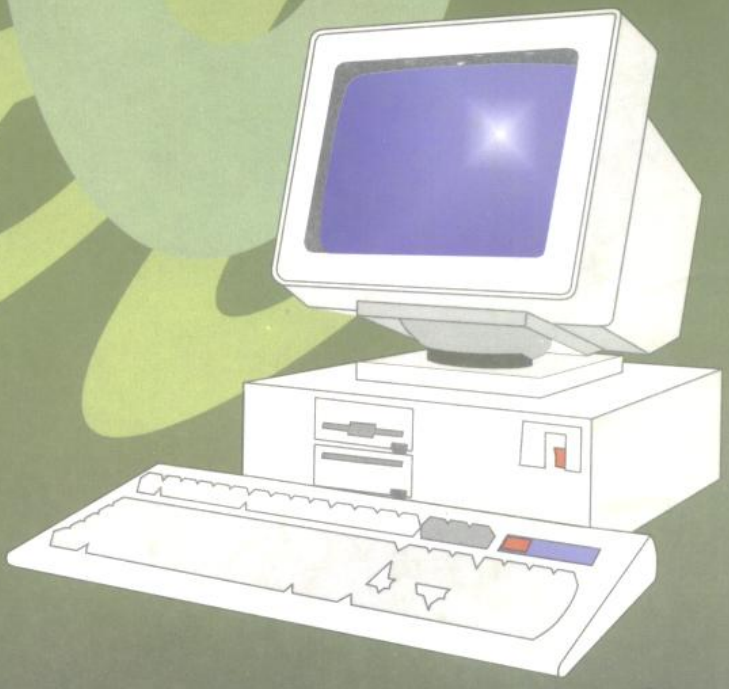
PC机汇编语言实战精解

PC JI HUIBIAN YUYAN SHIZHAN JINGJIE

李春生 编著



南开大学出版社

PC 机汇编语言实战精解

李春生 编著

南开大学出版社

内 容 简 介

这是一本颇具创新精神的汇编语言教材。作者完全打破了传统教材的谋篇布局，从“指令、算法与硬件相结合”这一基点出发，深入浅出地讲解了深奥难懂的汇编语言知识，在讲解中还融入了作者自己的实践体会。全书以大量的程序实例和上机实践为核心，详细地讨论了汇编语言的学习方法，很有启发意义。

适合于大中专院校学生与电脑爱好者作为入门导读之书。

图书在版编目（CIP）数据

PC 机汇编语言实战精解/李春生编著，—天津：南开大学出版社，1999.8
ISBN 7-310-01303-4

Ⅰ.P… Ⅱ.李… Ⅲ.微型计算机-汇编语言 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（1999）第 23483 号

出版发行 南开大学出版社
地址：天津市南开区卫津路 94 号
邮编：300071 电话：（022）23508542
出版人 张世甲
承 印 天津市宝坻县印刷厂印刷
经 销 全国各地新华书店
版 次 1999 年 10 月第 1 版
印 次 1999 年 10 月第 1 次印刷
开 本 787mm×1092mm 1/16
印 张 21.75
字 数 552 千字
印 数 1—3000
定 价 30.00 元

前 言

在所有程序设计语言中，汇编语言是最快速、最灵活且最有效率的一种，也是唯一能够发挥全部硬件潜能的语言。“汇编语言程序设计”历来是大中专院校计算机软、硬件及应用专业一门十分重要的必修课，学好汇编语言是掌握计算机原理、操作系统等课程的先决条件。近年来，随着各专业的相互融合，一些非计算机专业（如机电专业）也要求掌握计算机原理及汇编语言。可见“汇编语言程序设计”这门技术的应用是非常广泛的。

然而由于汇编语言是一种完全面向硬件的语言，因而与 C 语言、BASIC 语言等高级程序设计语言相比其掌握难度很大。目前普遍使用的一些教材及资料都是按照“CPU 结构 — 指令系统 — 算法 — 硬件”这样一种结构编写的，采用这种结构固然有构思严谨、逻辑清晰的优点，但它的一个显著的缺陷就是难点过于集中。由于把大量的指令集中于一章而将大量的程序例集中于另一章，这样势必会割裂指令、原理及算法的有机结合，因而使人学习起来感到困难。对于自学者而言，这个问题尤其突出。

本书的组织特点恰好避免了传统结构的弊端。笔者将最复杂难学的指令系统打散，将其分散到各章之中。全书主要以 IBM-PC 型计算机的软、硬件原理为主线，将指令、原理与算法结合在一起加以讨论，彻底解决了难点集中的问题。同时，书中还给出了大量实际运行过的程序例，并重点讨论了如何在计算机上调试、分析汇编程序的方法。这样不仅使学习汇编语言变得容易，同时可以使读者掌握上机操作和程序调试技术，因而本书具有很大的实用价值。

在写作风格上笔者采用了一种授课式的方法，每一章都如同一段讲义，这使得本书的语言比较通俗易懂。当然，专业的术语总是难以避免的，不过对于一些重要的概念，笔者都作了简洁的说明。

由于上述这些特点，因而本书非常适合于大中专院校工科专业的学生阅读，尤其适合于非计算机专业的学生以及自学计算机技术的电脑爱好者使用。

凡是笔者自认为比较重要的内容，在书中均以楷体字出现；带有  图标的内容希望读者能够认真阅读；带有  图标的内容希望读者仔细记忆。

这本书是对笔者自学过程的一个总结。由于作者经验与水平均十分有限，因此书中难免会有疏忽、错漏之处，恳请读者批评指正。

在本书的出版过程中，有多位编辑为此书的面市付出了辛勤的汗水，在此特向他们致以真诚的谢意，感谢他们对一个初出茅庐的作者给予的支持。

谨以此书献给我的双亲，为了回报他们多年的呵护与关怀。

李春生

1999 年春于北京

目 录

第 1 章 汇编语言基础知识 1
1.1 汇编语言的特点 1
1.2 汇编语言中的数 2
1.3 数学运算和逻辑操作 3
1.4 汇编语言中的文字和符号 6
1.5 数据的存储 8
1.6 寄存器 9
第 2 章 开始设计程序 11
2.1 如何发出声音 11
2.1.1 喇叭的构造 11
2.1.2 汇编伴侣——DEBUG.EXE 11
2.1.3 细看 PC 机 12
2.2 编制第一个程序 14
2.2.1 程序的输入和保存 14
2.2.2 程序透析 16
2.2.3 程序的缺点 20
2.3 回到 DOS 的怀抱 20
2.4 其它的改进方法 24
第 3 章 中断调用与子程序 27
3.1 中断的概念和处理过程 27
3.1.1 中断的基本概念 27
3.1.2 中断的处理过程 28
3.2 几个常用的软件中断 30
3.3 子程序 36
3.4 细致地咀嚼 41
第 4 章 奇妙的声音 45
4.1 控制定时器 45
4.1.1 定时电路 45
4.1.2 编制源程序 47
4.2 精确定时 57
4.3 数字的表示 66
4.4 更多的技术 68
第 5 章 子过程和串处理 80
5.1 子过程参数的传送 80
5.1.1 通过寄存器传送数据 80
5.1.2 通过堆栈传递参数 83
5.1.3 通过地址表传递参数 88
5.2 特殊的过程调用 91
5.3 数据串处理 97
第 6 章 文字输出与键盘输入 116
6.1 基础知识 116
6.1.1 认识显示系统 116
6.1.2 显示卡的类别 117
6.1.3 文字与图形的构成 117
6.1.4 显示模式 119
6.2 Video BIOS 的应用 120
6.3 直接操作 Video RAM 141
6.4 端口编程 152
6.5 键盘输入 159
第 7 章 文件控制块 168
7.1 文件操作基础 168
7.1.1 文件的概念 168
7.1.2 操作系统 168
7.1.3 处理的对象 168
7.1.4 两种方法 168
7.1.5 文本与二进制 169
7.1.6 文件处理的步骤 169
7.2 文件控制块 170
7.2.1 文件控制块的结构 170
7.2.2 文件的属性 170
7.2.3 驱动器号 171
7.2.4 记录块与记录 171
7.3 FCB 记录的随机存取 192
7.3.1 随机读写 192
7.3.2 随机块读写功能 195
7.4 FCB 完结篇 198
第 8 章 文件句柄功能与磁盘 207
8.1 文件句柄功能调用 207
8.1.1 文件的存取 210
8.1.2 一些辅助功能 218
8.2 细看磁盘 227
第 9 章 图形显示 238

9.1 BIOS 的图形功能 238
9.2 直接写屏 263
9.3 端口编程 267
第 10 章 细节补充 269
10.1 BCD 运算 269
10.2 标志寄存器 275
10.3 条件转移指令 279
10.4 其它指令说明 281
第 11 章 更高级的技术 291
11.1 内存驻留程序设计初步 291
11.2 内存分配技术 307
11.3 高级汇编技术 314
11.4 模块化程序设计 321

附录 A 21H 中断功能调用 328
附录 B 10H 中断功能调用 332
附录 C HGC 单显仿真 CGA 程序 334
附录 D BIOS 数据区 336

参考文献

第 1 章　汇编语言基础知识

1.1　汇编语言的特点

所谓汇编语言，其实质就是机器语言的一个高级的形式。我们知道，机器语言是CPU唯一可以真正“理解”的语言，它是用一些由“0”和“1”两个数字组成的一组数字来表示的。例如：1011000000000001（意思是将数字1放入累加器）。

这样的一组数字非常难以理解和记忆，毕竟程序员不是一块 CPU。为了使程序设计人员能够很好地记忆这些机器指令，简化程序设计工作，技术人员将这些怪异的数字用一些取自人类语言的简短的文字符号来表示，于是就产生了汇编语言。这些简短的文字符号称为指令助记符。例如上面的那个机器指令用汇编语言表达出来，就是“MOV AL，1”。

同高级语言相比，汇编语言具有一些极其突出的特点：

①汇编语言是一种完全面向硬件的语言，这同 BASIC 语言、C 语言之类的高级语言截然不同。多数高级语言都是面向问题的，例如：如果需要在屏幕上显示一串文字时，我们可以直接应用 BASIC 语言中的 PRINT 语句，或用 C 语言中的 PRINTF 函数，这个问题就迎刃而解了。而使用汇编语言编程，解决这个问题的最终操作是“将这些文字的 ASCII 码写入显示缓冲存储器中”。可见，汇编语言将这个问题转化成了对硬件（显示缓冲存储器）的操作（写入）。这是汇编语言的一个极其突出的特点，也是汇编语言同高级语言的最显著的差别；

②同高级语言相比，汇编语言编写的程序结构十分紧凑，运行速度很快。汇编语言同机器指令直接对应，编译速度快，同时，CPU“理解”其“母语”的速度远高于“翻译”高级语言的速度。因此，汇编语言是所有程序设计语言中运行效率最高的。这是汇编语言的一个最为突出的优点。当需要编写高速运行的软件时，例如编写图像处理程序，就往往使用汇编语言编写软件中的关键部分；

③用汇编语言编制程序十分费时，而且程序的质量直接受到程序员技术水平的影响，程序的可读性也很差。就像前面所举的输出文字的例子，用高级语言编程只需写一条语句，简单明了，极其直观。而用汇编语言编程则需写出一系列指令，这些指令都是些对硬件的操作，同“文字输出”这个问题没有明显的直接联系，因此程序的可读性很差。

④由于汇编语言是面向硬件的，所以用汇编语言编制的程序可移植性很差。显而易见，不同的 CPU 都有相互独立的指令系统，相互间无任何关系，就算是使用同一系列 CPU 的机器，因其外围硬件可能有差别，这也会使相同的程序在不同的机器上无法通用。

不难看出，汇编语言存在很多的弱点，但由于它具有一些高级语言所不具备的突出优点，所以汇编语言的应用范围还是很广的。特别是当用户需要研究计算机具体的工作原理的时候，还必须掌握汇编语言。

1.2 汇编语言中的数

高级语言中也多用十进制数。十进制数由0～9十个数字组合而成，逢10进1。但由于汇编语言是面向硬件的，因此，在汇编语言中使用的数字就是和硬件结合紧密的二进制数。

二进制数由0和1两个数字组成，逢2进1，也就是说，在十进制数中计算1+1时将得到一位数字的结果——2，而在二进制数中计算时将得到一个两位二进制数——10，表1-1列出了四位二进制数与十进制数间的对应关系。

在十进制数字中还有“数位”之分，个位，十位，百位，……在二进制数中也分各个数位。以4位二进制数1010为例，从右数第一位，称为bit0，第二位称bit1，以此类推。因此，1010的bit3和bit1位是1，bit2和bit0位是0。与十进制数一样，二进制数自右向左数位逐渐升高。最左端的数位为最高位，1010的最高位为1。

表1－1 4位二进制数与十进制数对照表

二进制数	0000	0001	0010	0011	0100	0101	0110	0111
十进制数	0	1	2	3	4	5	6	7
二进制数	1000	1001	1010	1011	1100	1101	1110	1111
十进制数	8	9	10	11	12	13	14	15

我们知道，计算机利用电路来记忆 1 和 0，那么 1 和 0 究竟和电路的工作状态有什么关系呢？通俗地讲，1 和 0 反映了电路输出（入）电压的高和低。如果电路输出（入）的电压高，比如达到了电源电压的幅度，此时电路的输出（入）就为 1。如果电路输出（入）的电压很低，比如接近 0V，那么这时电路的输出（入）就为 0。

二进制数字在实际应用中还有一些缺点，比如不便于书写，难于记忆等。为此，在汇编语言中还常用另一种数制——十六进制。十六进制数字由十六个数组成，逢 16 进 1。前 10 个数字和十进制一样，是 0～9，后 6 个数字用英文字母 A～F 来表示。表 1－2 列出了十六进制与十进制数间的对应关系。

表1－2 十六进制数与十进制数对照表

十进制	0	1	2	3	4	5	6	7
十六进制	0	1	2	3	4	5	6	7
十进制	8	9	10	11	12	13	14	15
十六进制	8	9	A	B	C	D	E	F

从表 1-2 可以看出，有些十六进制数与十进制数是一样的，区分不开。这个问题对于 1 位十六进制数而言倒还没有什么关系，可对于多位十六进制数来说就有些麻烦了。比如给出一个数“79”，它是十进制的“79”还是十六进制的“79（相当于十进制 121)”呢？所以为区分十六进制数与十进制数，特别规定了十六进制数尾部应加字母“H”(Hexadecimal)。十六进制数 79 应该写成“79H”才对。

对照表 1－1 和表 1－2，可以看出每个 4 位二进制数都有相应的十六进制数与之对应。若用十六进制数代替二进制数，在书写时可以使数位减少，简化了书写。同时也可以看到，十六进制数同二进制数之间没有实质上的差别，只是两种不同的表达形式而已。下面的例子介绍了二－十－十六进制数字间的转换方法，请大家自己总结其中的规律。

例 1.1　将十进制数 83 转换成二进制数。

解：采用短除法计算

```
2| 83      ------余数：1 ←——bit0
 2| 41     ------余数：1 ←——bit1
  2| 20    ------余数：0 ←——bit2
   2| 10   ------余数：0 ←——bit3
    2| 5   ------余数：1 ←——bit4
     2| 2  ------余数：0 ←——bit5
      1  ←————————————bit6
```

计算结果为 7 位二进制数：1010011。

例 1.2　将 8 位二进制数 10110110 转换成十进制数。

解：8 位二进制数各个数位的权[①]如下所示：

bit	7	6	5	4	3	2	1	0
权	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

$$\therefore\ 10110110 = 1\times2^7+0\times2^6+1\times2^5+1\times2^4+0\times2^3+1\times2^2+1\times2^1+0\times2^0$$
$$= 182$$

例 1.3　将十进制数 34567 转换成十六进制数。

解：仿照例 1.1 采用短除法计算

```
16| 34567     ------余数：7 ←—— 第 0 位
 16| 2160     ------余数：0 ←—— 第 1 位
  16| 135     ------余数：7 ←—— 第 2 位
     8  ←———————————————— 第 3 位
```

结果是一个 4 位十六进制数：8707。

例 1.4　将 16 位二进制数 1101101001100011 转换成十六进制数。

解：将此二进制数按每 4 位为一组分成 4 组

```
1101 1010 1100 0011
 ↓    ↓    ↓    ↓
 D    A    C    3
```

由表 1-1、表 1-2 查得

$$\therefore\ 1101101011000011 = DAC3$$

1.3　数学运算和逻辑操作

同十进制数一样，二进制数同样有相应的加、减、乘、除之类的运算，这些运算称为数学运算。由于数学运算还涉及到二进制的其它一些重要的知识，因此在这里暂时不做深入讨论。现在主要来讨论二进制数的逻辑操作。

这里所说的逻辑不同于广义的逻辑，事实上在计算机中的逻辑关系十分简单，只有四种

① “权”这个字最早指秤砣，同样一个秤砣在秤杆的不同位置可以表示不同的份量。引申到数字中，同样一个数在不同的数位上所表示的大小也是不一样的。

——与逻辑，或逻辑，非逻辑和异或逻辑。与、或和非的关系可以通过一个电路的例子来说明，见图1－1。

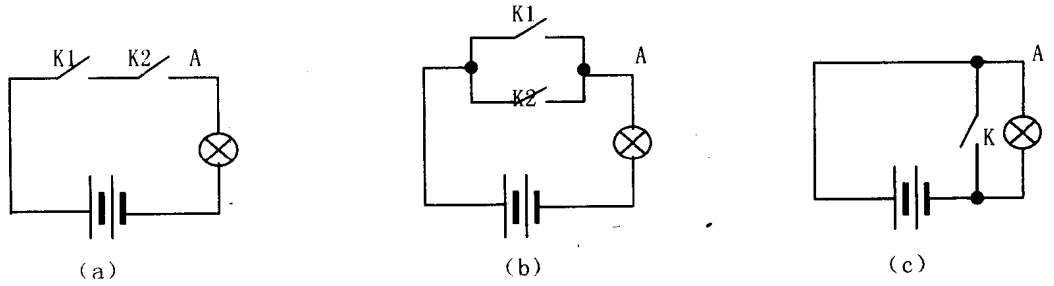


图1-1 三种逻辑关系示意图

在三个图中，灯被点亮的条件是什么呢？很明显，当A点电压和电源电压一致时（即A点输出为1时），灯就会亮。看来主要的问题就是如何使A点输出1。

对于（a）图，只有当开关K1和K2都闭合时，A点才会与电源接通，此时灯亮。若把“开关闭合”这一动作用“1”表示，把“开关断开”用“0”表示，则可以说，在（a）图中只有两个开关都是“1”时，A点才会输出“1”。这种开关状态与输出之间的关系就是“与”逻辑关系。

对于（b）图来讲，两个开关或者K1为“1”（接通），或者K2为“1”，或者两者都为“1”，均可以使A点输出为“1”，这两个开关与输出之间的逻辑关系就称为“或”逻辑关系。

对于（c）图而言，当K为“0”时A点才会输出“1”，K为“1”时电源被短路，此时A点输出“0”。这种逻辑关系称为“非”逻辑关系。

“异或”关系不大好用图表达，但是异或关系有一个重要的特点，就是当进行异或操作的两个数“相同”时所得结果就是“0”，而两个数“不同”时就得“1”。这是一个十分重要的特性，大家需牢牢记住。

所谓逻辑操作，就是把两个数按照选定的某种逻辑关系加以处理并得出结果的过程。逻辑操作通常用于使一个二进制数中的某些数位的状态变成我们需要的其它状态，而不改变其它位。

在汇编语言中，基本的逻辑操作有四种：与操作、或操作、非操作和异或操作。分别记作AND、OR、NOT和XOR。表1-3给出了这四种操作的具体情况。

表1－3 四种逻辑操作执行的结果

进行逻辑操作的两个数值		不同的逻辑操作及其结果			
A	B	AND	OR	NOT*	XOR
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

* 注：“非”操作只对一个数进行，表中选择的是A。

下面的例子说明了这四种逻辑操作的应用。

例1.2 给定一个8位二进制数10110100，

① 求一个8位二进制数，与给定的数作OR操作，要求结果为10111101。

② 求一个8位二进制数，与给定的数作AND操作，要求结果为00110000。

③ 对给定的数作什么操作可得到二进制数 01001011？

④ 若把给定的数同 00100010 作 XOR 操作，将得到什么结果？

解：① 将 OR 操作的结果同给定的数相比，不难发现只要把给定数字的 bit0，bit3 位置成 1，其它位状态保持不变，即可得出结果。因此可很容易求出 8 位二进制数 00001001，并可验证：

```
    10110100
OR) 00001001
    10111101
```

可见，OR 操作可以方便地将某个二进制数的特定位置成“1”而保存其它位不变。只要取另一个二进制数，让这个数中的相应数位——即和给定的数中要改变的数位相对应的位为“1”，而其它位为“0”，即可达到目的。

②将 AND 操作的结果同给定的数相比，可以看出只要把给定数字的 bit2、bit7 位置成 0，其它位保持不变，即得出结果。因此可很容易求出 8 位二进制数 01111011，并且可以验证：

```
    10110100
AND) 01111011
    00110000
```

可见，和 OR 操作相对应，AND 操作可以把某个二进制数的特定位置“0”而保持其它位不变。只要取另一个二进制数，让这个数的相应位为“0”，而其它位为“1”，即可达到目的。

③比较结果和给定数，可看出将已知数所有位取相反状态，可得到结果。因此可用 NOT 操作，即

```
NOT) 10110100
    01001011
```

④ 将两个数作 XOR 操作

```
    10110100
XOR) 00100010
    10010110
```

把得到的结果 10010110 同已知数相比较，可以看出只要将已知数的 bit1、bit5 两位取反，就能得出结果。将③和④进行比较，可以发现这样一个规律：NOT 操作可以将给定数的所有位取反，而 XOR 操作可以将给定数的特定位取反；进一步分析④不难看出，若把所得到的结果 10010110 和 00100010 再作一次 XOR 操作：

```
    10010110
XOR) 00100010
    10110100
```

又能得到已知的数。即取反后的数位又重新恢复原状态。因此我们说，XOR 操作可以反复改变给定数中的特定位状态。

1.4 汇编语言中的文字和符号

习惯上通常把文字和符号统称为“字符”。字符在机器中是如何表示和存储的呢？我们知道计算机可以方便地处理数字，因此如果把字符用数字来表示，就可以很方便地在计算机中存储和处理。所以在计算机中一般采用ASCII（美国标准信息交换代码American Standard Code for Information Interchange）码来表示字符。

计算机中所用的字符包括：

字母：A、B、…、Z，a、b、…、z；

数字：0、1、…、9；

专用符号：+、-、*、/、=、…

控制符号：CR（Carriage Return 回车）、LF（Line Feed 换行）…

这些字符的ASCII码都列在表1-4和表1-5中。

表1-4 ASCII基本字符对照表

ASCII值	字符*	ASCII值	字符	ASCII值	字符	ASCII值	字符
000	NULL	020	SPACE	040	@	060	`
001	SOH	021	!	041	A	061	a
002	STX	022	"	042	B	062	b
003	ETX	023	#	043	C	063	c
004	EOT	024	$	044	D	064	d
005	END	025	%	045	E	065	e
006	ACK	026	&	046	F	066	f
007	BEL	027	'	047	G	067	g
008	BS	028	(	048	H	068	h
009	HT	029	)	049	I	069	i
00A	LF	02A	*	04A	J	06A	j
00B	VT	02B	+	04B	K	06B	k
00C	FF	02C	,	04C	L	06C	l
00D	CR	02D	-	04D	M	06D	m
00E	SO	02E	。	04E	N	06E	n
00F	SI	02F	/	04F	O	06F	o
010	DLE	030	0	050	P	070	p
011	DC1	031	1	051	Q	071	q
012	DC2	032	2	052	R	072	r
013	DC3	033	3	053	S	073	s
014	DC4	034	4	054	T	074	t
015	NAK	035	5	055	U	075	u
016	SYN	036	6	056	V	076	v
017	ETB	037	7	057	W	077	w
018	CAN	038	8	058	X	078	x
019	EM	039	9	059	Y	079	y
01A	SUB	03A	:	05A	Z	07A	z
01B	ESC	03B	;	05B	[	07B	{
01C	FS	03C	<	05C	\	07C	|

续表

ASCII值	字符*	ASCII值	字符	ASCII值	字符	ASCII值	字符
01D	GS	03D	=	05D	]	07D	}
01E	RS	03E	>	05E	^	07E	~
01F	US	03F	?	05F	_	07F	

* 注：000H-01FH 本为控制码，但在 PC 机屏幕上有时也会显示出特殊字形，比如 00DH 是控制码“CR（回车）”，直接写入屏幕时会显示出“♪”。

表1-5 ASCII扩展字符对照表

ASCII值	字符	ASCII值	字符	ASCII值	字符	ASCII值	字符
080		0A0	á	0C0	└	0E0	α
081	ü	0A1	í	0C1	┴	0E1	β
082	é	0A2	ó	0C2	┬	0E2	Γ
083	à	0A3	ú	0C3	├	0E3	Π
084	a	0A4	n	0C4	─	0E4	Σ
085	à	0A5	N	0C5	┼	0E5	σ
086	a	0A6	a	0C6	╞	0E6	μ
087		0A7	o	0C7	╟	0E7	τ
088	ê	0A8	¿	0C8	╚	0E8	Φ
089	è	0A9	⌐	0C9	╔	0E9	θ
08A	e	0AA	¬	0CA	╩	0EA	Ω
08B		0AB	1/2	0CB	╦	0EB	δ
08C		0AC	1/4	0CC	╠	0EC	∞
08D	ì	0AD		0CD	═	0ED	ϕ
08E	A	0AE	«	0CE	╬	0EE	∈
08F	A	0AF	»	0CF	╧	0EF	∩
090	E	0B0	▒	0D0	╨	0F0	≡
091		0B1		0D1	╤	0F1	±
092		0B2	▓	0D2	╥	0F2	≥
093		0B3	│	0D3	╙	0F3	≤
094		0B4	┤	0D4	╘	0F4	⌠
095	ò	0B5	╡	0D5	╒	0F5	⌡
096		0B6	╢	0D6	╓	0F6	÷
097	ù	0B7	╖	0D7	╫	0F7	≈
098		0B8	╕	0D8	╪	0F8	°
099		0B9	╣	0D9	┘	0F9	•
09A		0BA	║	0DA	┌	0FA	·
09B		0BB	╗	0DB	▉	0FB	√
09C	£	0BC	╝	0DC	▄	0FC	n
09D	¥	0BD	╜	0DD	▌	0FD	Z
09E	Pt	0BE	╛	0DE	▐	0FE	■
09F	ƒ	0BF	┐	0DF	▀	0FF	BLANK

用 ASCII 码表示字符可以很方便地由计算机处理，也便于在机器之间交换文字信息。当我们在键盘上按下字母“a”时，计算机究竟收到怎样的信息呢？实际上主机从键盘收到的信息就是一个二进制数 01100001。由于字符与数字之间的对应关系在 ASCII 码表中规定死了，计算机当然知道键盘传输来的这个数字意味着什么，所以它会将这个数重新转换为人所能看懂的形式将并其显示在屏幕上。因此“键盘输入—主机—显示输出”的过程实际上就是“文字—数字—文字”的转换过程，这种转换由计算机自己完成。

注意，标准的ASCII码表内只有128个字符和控制码，用一个7位二进制数就可以表示。不过我们实际使用的都是8位二进制数，最高位用作校验位。在PC电脑中，ASCII码表被扩展了，最高位不再用于校验，这样一来就多出了128个字符，这128个字符通常称为“扩展ASCII码”，我们平常所看到的表格线，以及“Σ”、“Ω”这样的字符都存于扩展ASCII码表中。

1.5 数据的存储

计算机之所以被广泛应用，主要原因就在于它能够“记忆”。即将数据存储在它的“存储器”中。那么数据究竟是以何种形式存于存储器中的呢？前面已经讨论过，1位二进制数只能表达0和1两种状态，用于表示数字也只能表达0和1两个数字。而我们平时应用的数的范围是很大的。因此，若数据在存储器中是按“位”存放，即CPU每次仅能从存储器中取得1bit数据进行处理，那么这样的计算机效率是极低的。所以，“位”（bit）并不是存储器中所使用的最小存储单元。

实际上，数据是按照每8个“位”为一组的形式存于存储器中，也就是说，CPU每次从存储器中取出或放入数据的最小宽度是8个bit。我们把这8个bit单独取了个名字——“字节”（BYTE）。所以说，存储器的最小单元就是“字节”。

有时我们嫌“字节”这个单位太小，因此在实际应用中还常用“千字节”（KB），“兆字节”（MB）和“千兆字节”（GB）。1KB＝1024B，1MB＝1024KB，1GB＝1024MB。①

不同的CPU所能配备的存储器的容量是不同的，例如8086/88最多可配备1MB存储器，而80286最多可配备16MB存储器。为什么有这样的差别呢？图1-4表示了CPU同存储器之间的联接形式。

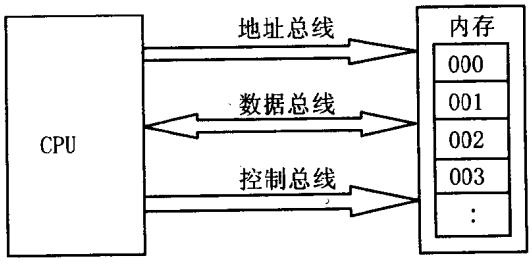


图1-4 存储器与CPU之间的信号传输

可以看到，存储器中的每个字节都有一个编号，这个编号在技术上称为存储单元的“地址”（Address）。这就像是生活中的门牌号码一样。假如要把一封信邮寄到收信人手中，势必要写出收信人的地址。同样，CPU要想把数据发到存储器的某个单元中，也要给出这个存储单元的“地址”。因此，在CPU和存储器之间就有一组专门传送“地址”的线路，这组线路称为“地址总线”。

地址总线不是一根，而是一组。每根地址线都有0或1两种状态。这一系列的0和1组成一个二进制数。当CPU把这个二进制数传到存储器后，哪个存储单元的编号恰好和这个数

① 这里所说的“千”、“兆”指的分别是2^{10}、2^{20}，不是一般意义上的10^3、10^6。

相等，则这个存储单元就被CPU“选中”，CPU就可以把这个存储单元所存的数据通过另一组线路（数据总线）取出，或向这个存储单元存入新的数据。

不同的CPU所具有的地址线数量是不一样的，像8086/88只有20根地址线，所能给出的地址范围是0～（$2^{20}-1$）（1048575），所以8086/88只能配备1MB存储器。而286有24根地址线，当然可配备2^{24}个字节的存储器。386/486有32根地址线，所以最多可配备4GB存储器。

前面所说的存储器都是直接和CPU相联接的，这些存储器和CPU往往安装在同一块电路板上，因此通常把这类存储器称为“内存储器”（内存）。在机器中还有另一类存储器——“外存储器”，这类存储器一般指的是软盘、硬盘、磁带和光盘等容量较大的存储设备。

外存储器和内存相联接而不和CPU直接联接，即CPU无法直接将外存中的数据取到自己内部进行处理，而只能先通过另一部分控制电路将外存中的数据取到内存中，再进行进一步的处理。处理后的数据也是先放在内存中，再传至外存。

正是由于这个原因，所以外存的容量并不受CPU地址线数量的限制。其容量可从几百KB直至几个GB不等。而且外存往往是机械与电子结合的设备，其数据存取速度远低于内存。

1.6 寄存器

寄存器又分为内部寄存器与外部寄存器。所谓内部寄存器，其实也是一些小的存储单元，也能存储数据。但同存储器相比，寄存器又有自己独有的特点：

①寄存器位于CPU内部，数量很少，仅14个；

②寄存器所能存储的数据不一定是8bit，有一些寄存器可以存储16bit数据，对于386/486处理器中的一些寄存器则能存储32bit数据；

③每个内部寄存器都有一个名字，而没有类似存储器的地址编号。

寄存器的功能十分重要。CPU对存储器中的数据进行处理时，往往先把数据取到内部寄存器中，而后再作处理。关于各个寄存器的具体功能后面会深入讨论。

外部寄存器是计算机中其它一些部件上用于暂存数据的寄存器，它与CPU之间通过“端口”交换数据，所以外部寄存器具有寄存器和内存储器双重特点。有些时候把外部寄存器称为“端口”，这种说法不太严格，但经常这样说。

外部寄存器虽然也用于存放数据，但是它保存的数据具有特殊的用途。某些寄存器中各个位的0、1状态反映了外部设备的工作状态或方式；还有一些寄存器中的各个位可对外部设备进行控制；也有一些端口作为CPU同外部设备交换数据的通路。所以说，端口是CPU和外设间的联系桥梁。

CPU对端口的访问也是依据端口的“编号”（地址），这一点又和访问存储器一样。不过考虑到机器所联接的外设数量并不多，所以在设计机器的时候仅安排了1024个端口地址，端口地址范围为0～3FFH。图1-5反映了CPU、内存、端口和外设间的联接关系。

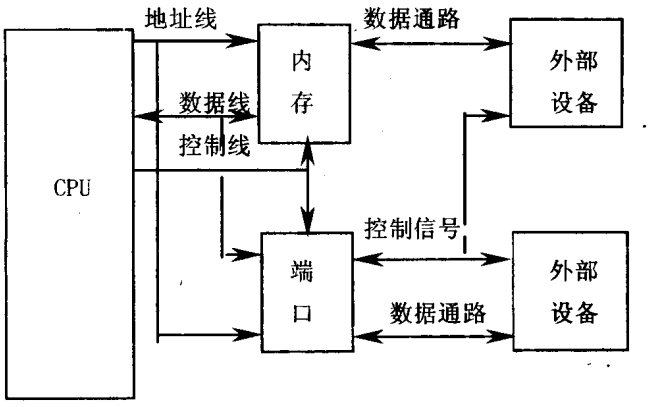


图1-5　CPU、内存、端口和外设的联接关系

本章结束语

汇编语言是面向硬件的语言，它所真正面对的就是存储器、寄存器和端口。而它所完成的工作无非是在内存中保存数据，在寄存器中“加工”数据，通过端口传输数据或控制设备。这一特点随着后文对汇编语言更深入的探讨就会逐渐显露出来。

汇编语言中的每一条指令都会控制电脑完成一个细微的动作，这些细微的动作组合在一起就会产生一种宏观的效果。CPU 的一举一动都在程序的精确控制之下完成。这样的能力可不是高级语言所能拥有的。这也正是汇编语言能够吸引人的地方。

第 2 章　开始设计程序

前一章提到汇编语言的一个用途是通过端口控制外部设备，本章我们就要用汇编语言编制一些小程序来控制机器中最富于趣味性的一个小设备——喇叭。我们的程序将使这个小部件发出各种各样的声音，从简单的嘀嘀声直至叮叮咚咚的音乐，还有乒乒乓乓的枪声。同时我们也会学到许多指令和程序设计技术。这对于我们今后的学习有很大的帮助。好，下面就让我们带着好奇的心情开始这艰难的开端吧。

2.1　如何发出声音

2.1.1　喇叭的构造

喇叭的构造如图 2—1 所示，它主要由纸盆、线圈、永磁铁等组成。当有交变电流流过线圈时，线圈产生的磁场将和永磁铁的磁场相互作用，使线圈产生振动，从而和线圈相连的纸盆也会随之振动，产生声音。

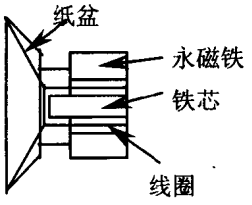


图2-1　喇叭的构造

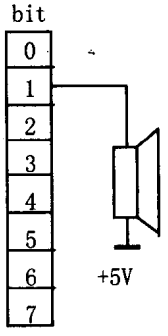


图2-2　喇叭与机器的连接

那么 PC 机中的小喇叭是怎么与机器相联的呢？我们能否改变流过喇叭线圈的电流呢？图 2—2 表示了喇叭与机器间简单的联接情况（实际情况要复杂些）。喇叭的一端接在电源正极上，另一端与机器中的 61H 端口的 bit1 位相联。可以想象，若能连续改变 61H 端口的 bit1 位的 0、1 状态，就可以使喇叭线圈内的电流时有时无，从而使喇叭发出声音。我们编制的汇编程序的工作，就是连续改变 61H 端口的 bit1 位状态。

2.1.2　汇编伴侣—— DEBUG. EXE

如何将汇编程序输入计算机？各种高级语言都有自己的一套解释或编译程序，汇编语言也同样有自己的编译器。然而我们现在还没有良好的基础，因此现在就开始介绍汇编语言的编译系统还有些为时过早。好在 DOS 给我们提供的一个“迷你”汇编器——DEBUG。