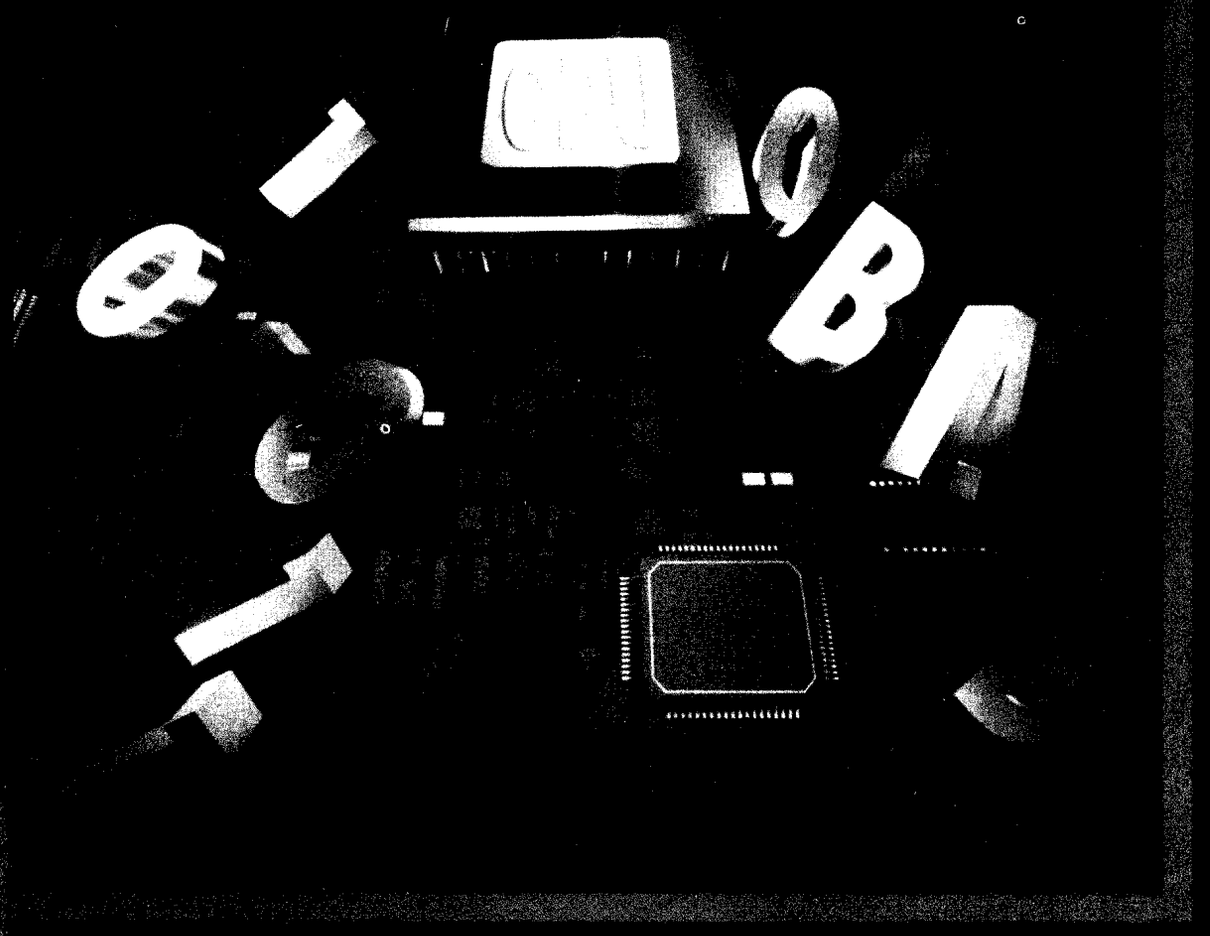
第3波

蔡友家 著

MASM

汇编语言应用 6.11



13

机械工业出版社

本书从汇编语言的基础知识入手，深入浅出地向您介绍最新版的MASM 6.11，使您逐步掌握有关MASM 6.11的各种知识。本书第1～2章为基础知识部分；第3～5章介绍了如何使用和处理汇编语言程序、数据类型以及地址与指针的重要概念；第6章介绍了重要的中断概念及MASM中常用的宏；第7～9章是有关指令、程序库和子程序的知识；第10章介绍了有关磁盘的操作和文件的处理，无论您是初学者还是汇编语言的使用高手，本书都能使您从中获益。

本书繁体字版名为《MASM 6.11组合语言应用》，由第3波文化事业股份有限公司出版，版权归第3波文化事业股份有限公司所有。本书简体字中文版由第3波文化事业股份有限公司依出版授权合同约定，授权机械工业出版社出版，未经出版者书面许可，本书的任何部分均不得以任何形式或手段复制或传播。

图字：01—96—1076

图书在版编目（CIP）数据

MASM 6.11 汇编语言应用/蔡友家著．—北京：机械工业出版社，1998.2
ISBN 7-111-06088-1

Ⅰ.M… Ⅱ.蔡… Ⅲ.汇编语言，MASM6.11 Ⅳ.TP312

中国版本图书馆CIP数据核字（97）第29260号

出 版 人：马九荣（北京市百万庄大街22号 邮政编码100037）
责任编辑：王冠宇 版式设计：王 颖 责任校对：林去菲
封面设计：姚学峰 责任印制：路琳

中国建筑工业出版社密云印刷厂印刷·新华书店北京发行所发行

1998年5月第1版第1次印刷
787mm×1092 mm$^{1}/_{16}$·17印张·415千字
0 001—3 000册
定价：30.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

目　录

此为试读，需要完整PDF请访问：www.ertongbook.com

第 1 章　汇编语言介绍

在这章中您将学一些编写汇编语言的基础。内容包括处理器（Processor）、计算机语言（Language）、数制（Number System）及软件开发工具（Software Development Tool）。

对大部分的计算机而言，有一个非正式的说法，称计算机的心脏是处理器（Processor），你可以在主机板上找到它。在 IBM 兼容计算机上使用的是 Intel X86 家族的处理器。从发明到最近，处理器均是以数字来命名，如 8088、8086、186、286、386、486 等（数字越大代表功能越强）。目前最新、功能最强的处理器，并不是以数字命名，而是称为 Pentium，接下来就是 P6 了。

1.1　机器语言与汇编语言

1.1.1　机器语言

处理器是由许多复杂的芯片组合而成。然而，就我们用户观点来看，所有的处理器只是一个接一个去执行简单指令的机器。

处理器所能执行的指令集合称为指令集，不同的处理器指令集的大小也有所不同。例如 486 处理器指令集就包含 206 条基本指令。而我们在操作系统下，执行一个简单的 BASIC 程序，必须被分解成许多处理器能执行的步骤。

按照指令使用的规则，处理器能直接执行的指令集合称为机器语言。在机器语言中每条指令都是一组数字。一个机器语言程序可以说是由一些非常长的数字字符串组合而成。现在人们已不再直接用机器语言编写程序，而采用较高级的计算机语言，以节省设计程序的时间及工作量。

广义上，我们将计算机语言分为：

（1）高级语言（如 C、C＋＋、BASIC）。

（2）低级语言（汇编语言）。事实上，真正的计算机语言是机器语言（也就是计算机真正能看懂、能执行的语言）。使用低级语言编写的程序需通过编译程序编译成机器语言。

1.1.2　机器指令

一个机器指令当然是一个二进制码数字（也就是只有 0 或 1）。机器指令直接对 CPU 下命令，去执行某一个操作，所以它只对 CPU 有意义。

机器语言指令集只对某一特定的 CPU 有用。IBM-PC 兼容计算机使用 Intel 系列的 CPU。而在其它指令集，如大机器上的机器语言指令集可能就与 PC 所使用的机器语言指令集不同。

例 1：一个典型的双字节 IBM-PC 机器指令如下所示，第 1 个字节 B0 是操作码（op code），第 2 个字节 05 是操数。

```
B0      05   ;  MOV  AL，05
↑       ↑
操作码  操作数
```

例 2：B4 05；mov AH，05

例 1 与例 2 的操作数均相同，只有操作码不同。可知不同的机器指令会有不同的操作码。

1.1.3 汇编语言

编译程序最大的用处就是将我们所写的汇编语言程序读入 RAM 中，并产生一个与机器语言对等的程序。用高级语言写程序时，除了要了解语言的指令外，还需要知道如何使用它去产生你需要的结果，而编写汇编语言程序要学的地方则更多。除了上述内容之外，还必须了解计算机的基本构造以及数据如何存储与使用。

表 1-1 为几种处理器的指令集大小：

表 1-1 指令集大小

处 理 器	指令集大小(条)
8088/8086	115
186	126
286	142
386	200
486	206
Pentium	216

汇编语言能帮你解开硬件与软件的奥秘。想要了解计算机硬件与操作系统 Operating System)、AP (Application Programs) 如何与操作系统之间工作，就非得从汇编语言下手不可。

每一种计算机及其家族都使用不同的机器指令和汇编语言。本书所介绍的程序均可在 IBM PC、PC/XT、PC/AT、80286、80386、80486 上执行。汇编语言最接近机器语言，也是最接近计算机硬件的语言。在许多要求速度的场合，汇编语言是第一选择。

虽然高级语言能为我们带来许多便利，但同时也有许多限制，而汇编语言却没有这方面的困扰。但汇编语言的自由却需付出代价，需要程序员谨慎地处理许多细节。

有些高级语言所不能做到的工作，可将汇编语言写成一个子程序而直接调用此子程序替你工作。或许您用 C 或 Pascal 进行一些屏幕显示操作觉得不够快时，您可以利用汇编语言替你加快这项工作。

1.1.4 汇编语言指令

```
mov  AX，BX
```

上例中汇编语言指令意义是将寄存器 BX 的内容送到寄存器 AX 内。所以可看出 AX 是目的运算符，BX 是源运算符。执行完 mov AX，BX 指令后，BX 内容并不改变。

我们可以说每一条汇编语言指令都会对应或关联一条机器指令。而高级语言，如 C、Pascal 或 Basic，就须靠编译程序翻译成许多机器指令，效率上就差多了。

1.1.5 机器语言与汇编语言

大家都知道高级语言会将我们写的源程序(source code)通过编译程序翻译成机器懂的语言。而汇编语言也有编译程序 (Assembler) 将程序编译成机器语言，只不过我们用汇编语言写的指令在编译时效率会更高。

1.2 处理器与协处理器

除了处理器（Processor）外，在主机板上可能还有协处理器（Coprocessor）。例如数学协处理器可协助处理器执行数学运算，包含浮点数运算，而且执行速度相当快。虽然计算机可以用软件模拟，但却要牺牲一点时间。

486 和 Pentium 级的处理器一般都内置有数学浮点运算器（486 SX 没有）。使用数学浮点运算器是一个复杂的主题，并不在本书介绍范围内。表 1-2 为数学协处理器指令集大小。

表 1-2 数学协处理器指令集大小

处 理 器	数学协处理器	指令集大小（条）
8088/8088	8087	77
186	8087	77
286	80287	74
386	80387	80
486	内置	80
Pentium	内置	80

虽然 386、486 和 Pentium 比早期的处理器复杂，但用汇编语言设计程序却和在 8088 上一样。以前在 8088 上写的程序在较新的处理器上也同样能顺利执行。

早期的处理器（如 8088、8086、80186）有一个最大的限制是不能使用超过 1MB 内存。286 使用两种不同的模式来解决这个问题。

（1）实模式（Real mode） 执行时就和在 8088、8086、80186 一样，对所有在原始 8088 结构下的 DOS 程序完全兼容。

（2）保护模式（Protected mode） 有 3 个最重要的改进。

1）可使用 16MB 内存。

2）可使用外部磁盘空间来提供 1GB 的虚拟内存（Virtual memory）去模拟内存。

3）提供硬件多任务并行，能快速从一个程序切换到另一个程序。

386 有一最重要的突破就是增加了一个虚拟 86 模式（Virtual 86 mode），它可允许处理器在同一时间执行多个程序，且每一个程序就像在它自己的 8086 机器上一样。除此之外，386 在保护模式下可使用到 4GB 的真实内存（虽然真实硬件并没有提供至此）和 64TB 的虚拟内存。

如果使用一个特别的控制程序，如 Windows，你也可以使 386 在同一时间工作如同多重 DOS 一样。而目前最著名的多任务系统如 OS/2 和 Windows NT 也是使用虚拟 86 模式去提供内置的多重 DOS。

486 和 Pentium 用 cache controller（控制处理器内快速的内存）和内置的数学协处理器去汇编 386 的以上功能。从程序设计的观点来看，我们使用 486 和 Pentium 时就和 386 一样。8088 以后所增加的保护模式的额外指令，除非你要编写高级的系统软件，如操作系统、设备驱动程序或编译器，否则您将不必使用到这些指令。因此几乎所有在 Intel X86 处理器上编写

的程序就如同在 8088 上一样，特别是在编写 DOS 程序时。

本书的目的是让你对基本汇编语言有一个详细的了解。我们基本上不会使用应用在新的处理器上的指令。如果我们提到时将会指出为什么用新的指令会较好。

1.3 何时要使用汇编语言

因为高级语言比较简单，也具有概念性和可读性，所以在大部分工作中，高级语言是我们第一选择。但如果你碰上一些不能解决的事情时，就非要使用汇编语言不可了。因为它能直接控制处理器，所有在计算机上能做到的事情，汇编它都能做到。尤其在加速程序的执行时，你可以将程序中最浪费时间、最需要时间的地方改用汇编语言编写，再将它们连接在一起即可。

使用汇编语言的另外一个理由是，内存空间是昂贵的，而用汇编语言编写的程序所产生的可执行代码是最小的，这也是它速度快的原因之一。为了节省内存空间，使用汇编语言也是重要的方法之一。

对大部分的人而言，是不需要去学习或使用这样低级的语言，但对一个专业的程序员来说是必需的。当你在阅读计算机杂志时，其中的例子可能都是用汇编语言写成的，因为它比较容易解释处理器如何工作。如果不学习汇编语言的话，你将不能通过阅读去获得最新的信息和技巧，以成为一个最顶尖的程序员。

1.4 操作系统（Operating System）

计算机在执行时，操作系统是主要的控制程序。汇编语言程序员有两种情况下会使用到操作系统。

你必须对操作系统下达你要执行的命令，如 copy 文件，执行一个程序等等。在你的程序中，可能会利用到操作系统所提供的许多功能服务，如 I/O、存取文件或目录。像一些比较低级、复杂的工作，操作系统都会提供许多功能可以调用。

本书是假设你的操作系统是 DOS (Disk Operating System)。如果你是使用其它的操作系统如 OS/2 或 Windows NT，你还是可以使用这本书去学习汇编语言程序设计。如果你的程序有直接调用操作系统的功能时，其中的接口 (interface) 就需要你去查你的操作系统参考手册。

1.5 你需要的软件

要编写一个汇编语言程序，你需要有编辑器 (Editor)、编译程序 (assembler)、链接程序 (Linker)，以及调试器 (Debugger)，一般有两种选择。

1. 你可以使用一个全屏幕程序开发环境如 Programmer's WorkBench (PWB，由 Microsoft 提供) 或 Programmer's Platform (由 Borland 提供)。这种环境综合了你要建立程序或调试程序所需用到的工具。而你所要做的工作在下拉式窗口（或称菜单）中都可找到。

2. 使用文本编辑器（需以 ASCII 码存储），在 DOS 命令行下直接编译、链接、调试。在 DOS 中有 EDIT (5.0 版以后才有)、EDLIN 可进行程序的编辑，不必额外再花钱购买其它的

文本编辑器。

DOS 本身提供了一个叫 DEBUG 的调试工具。使用 DEBUG 的好处在于使用简单、自由以及占用较少的内存，但毫无界面可言。建议使用 MASM 所提供的 CodeView 去调试。

1.6 计算机的数制

1.6.1 位（Bit）和字节（Byte）

从计算机输出的数据是我们很容易看懂的文字、数字或符号。然而对计算机而言，它只不过是一个脉冲；如开与关的信号。而且这些信号所代表的意义都只有两种状态，我们常称为 0 与 1。每个单一的 0 或 1 称为位（bit），位也是计算机表示数据的最小单位。若将 n 个位串起来，就有 n 位，可形成 2^n 种不同的数字。在计算机内任何数据，如数字、字母、符号，都是利用这种位串的方式来表示。而 8 位组成一个字节（byte）如表 1-3 所示。而字（word）则须视不同的处理器而有不同的大小，在 80286 中，1 word＝16 bit。

表 1-3

位　数	名　称
8	byte
16	word（2 bytes）
32	double word（4 bytes）
64	quadword（8 bytes）
128	paragraph（16 bytes）

常见的计算机数制（如表 1-4）所示，有 Binary（二进制），Octal（八进制）、Decimal（十进制）、Hexadecimal（十六进制）。除了 Octal（八进制）在 PC 中较少见外，其余在本书中都将经常出现。每一种数制，均用一个基数（radix）来转换。人们常用的十进制，就是用基数 10 来表示数字。

表 1-4　常见的计算机数制

Base 基　数	可　能　的　数　字
Binary（2）	01
Octal（8）	01234567
Decimal（10）	0123456789
Hexadecimal（16）	0123456789ABCDEF

注：十六进制中　A 代表 10　B 代表 11　C 代表 12　D 代表 13　E 代表 14　F 代表 15

1.6.2 Binary Number

计算机中所储存的指令和任何数据均是采用一组二进制的数码表示。所以用 0 和 1 来代表数据与数字是最直接和最适合的。而二进制系统是以 2 为基数，所以只有 0 与 1 两个数字。

1100（Binary）＝12（Decimal）$=1*2^3+1*2^2+0*2^1+0*2^0$

此为试读，需要完整PDF请访问：www.ertongbook.com

一个数字称为1bit（0 or 1）。8个bit组成的位串（加10101010）即为一个字节（1B），这是所有计算机最基本的存储单位。1 byte就可代表一个指令、字符或数字。二进制转十进制方法如下：

$$1100\ 0011\ (B) = 1*2^7+1*2^6+0*2^5+0*2^4+0*2^3+0*2^2+1*2^1+1*2^0$$
$$= 195\ (D)$$

习惯上，如果是二进制数字，会在数字最右端加上一个B，代表是二进制数字，避免与十进制数字产生混淆。

1.6.3 Hexadecimal Number

对人而言，二进制并不易阅读，所以有以16为基数的十六进制表示法。前面10个用0到9表示，后6个用A到F表示。十六进制介绍如下：

0011	1100	0101	1001	(B)	(二进制)
3	C	5	9	(H)	(十六进制)

由上例可知，二进制转换为十六进制只须四位一组即可顺利转换。

习惯上，会在十六进制数字最右端加上一个H，代表是十六进制数字。

十六进制转十进制如下

$$1A5h = 1*16^2+10*16^1+5*16^0$$

如果你仔细看会发现，我们的计算机系统中，0代表第一个数字，而不是以1为第一个数字，1就变为第二个数字。

以1B表示一数字可发现，在二进制中11111111（255）为所能表达的最大数字，在十六进制中为0FFh（255）。

二进制加法

```
  0101
 +0011
------
  1000
```

※ 与十进制加法一样，从个位加起，由右往左，超过1，就进位到左边一位中，依此类推。

十六进制加法

```
  28A70
 +90B6
--------
  31B26
```

※ 与二进制相似，逢16便进位。

二进制减法

```
  0101
 -0011
------
  0010
```

※ 与十进制减法一样，若不够减则向前借位。

十六进制减法

```
  28A70
 -90B6
--------
  1F9BA
```

※ 与十进制减法一样，若不够减向前借位。

1.6.4 有符号数与无符号数

站在人的立场上，数字分为无符号数与有符号数。但计算机在处理数字时，仅将其视为一组 0 与 1 的组合，而并不知道数字是有符号数或无符号数。

1.6.4.1 无符号数

若用 1 个 byte 字节代表一个数字时，所有 8 个 bit 均可用来表示一数字，并没有负数，最小值为 0，最大值为 11111111B＝0FFh＝255。仔细观看可发现最大值为 2^8-1。由此可推知，如以两个 bytes 表示一个数字，则最大值为 0FFFFH＝$2^{16}-1$＝65535。

1.6.4.2 有符号数

有符号数的意思是数字分为正和负。以最高位来表示该数目的正负号，称为符号位(sign-bit)。如果该位数字大于或等于 0，则此数字的符号位必为 0；若该位数字小于 0，则符号位便为 1。

也就是说，如果该数字为 1byte，则只有 7 位（从 0 到 6）用来表示数据。如该数字为双字节，则只有 15 位（从 0 到 14）用来表示数据，以此类推。

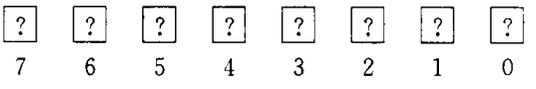
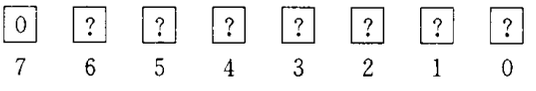
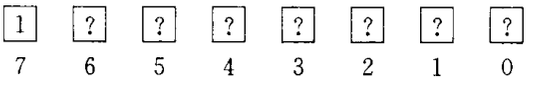


图 1-1 无正负号；所有位均用来表示数字



注：符号位为 0，表示正数



注：符号位为 1，表示负数

图 1-2 有正负号；只有较低的 7 位用来表示数字

在 IBM PC 系统中，负数的表示法是用 2 的补码（2′ s complement）来表示。二进制数要表示负数，只要将其相对的正数的二进制数字的所有的位变反的结果加 1 即可得到负数。

变反的意思是将原来为 1 的换成 0，原来为 0 的换成 1。

下面的例子可以让你很清楚地了解负数的转换过程。0 的二进制表示法为 00000000B，所以没有负 0。就算取补码为 11111111B 再加上 1 是 00000000B，还是为 0。

(1) 1 的二进制表示法为 00000001B

(2) 求反结果 11111110B

(3) 再加上 1 11111111B＝0FFH

可知－1 表示为 0FFH，而＋1 表示为 01H。我们可以再发现一个有趣的结果，如果我们将－1 与＋1 相加，则

```
  0000 0001        (1) 表示有进位，为进位位。
 +1111 1111
-----------
(1) 0000 0000
```

可知 1＋（－1）的结果为零，这与我们所想像的正好相符。

通过上述的讨论，表 1-5 可知 1 byte 可表示由＋127～－128 间的数字（整数）。若为 2 bytes 可代表＋32767～－32768。字节越多则表示范围愈大。

表 1-5

字　长	可代表的整数范围
1	＋127～－128　(2^7-1～-2^7)
2	＋32767～－32768　($2^{15}-1$～2^{15})
4	＋2147483647～－2147483648　($2^{31}-1$～2^{31})

1.6.4.3　内存大小的计算

计算机处理数据是以 byte 为单位，而不是以 bit 为单位。所以要说内存大小，应说成多少 byte。但 byte 还是太小，所以习惯上是以 kilobytes（KB）或 Megabytes（MB）来计算。

$$1\ \text{KB}=2^{10}\ \text{bytes}=1024\ \text{bytes}$$

$$1\ \text{MB}=2^{20}\ \text{bytes}=1048576\ \text{bytes}$$

例如我们会说基本内存为 640KB。主板上的 RAM 有 16MB。

1.6.4.4　BCD 码

BCD 码（Binary Code Decimal）是 IBM-PC 系统另一种数字表示法，这种码特别适合处理十进制数字，包含小数点。

所谓 BCD 码就是用四位表示一个十进制数字。而 BCD 码又可分为压缩式（Packed）和非压缩式（Unpacked）。

1　压缩式 BCD：一连串的四位为一组来表示一个十进制数字。如下：

```
0111 : 1001
  ↑     ↑
  7     9
```

由上例可知一个字节就可存储两个 BCD 数字。

2　非压缩式 BCD：每个字节只以最低四位来表示一个十进制数字；高四位便无定义，就不用它了。如下所示：

```
???? :1001     ???? :0011
    ↑              ↑
    9              3
```

由上例可知存储两个数字就需要两个字节，这比压缩 BCD 码浪费空间。

用 BCD 码代表数字作算术运算时，程序员须作特别的笔记工作，因为计算机均是以二进制码运算，并不知道它是一个 BCD 数字。

1.7　字符存储格式

计算机是用 0 与 1 组成的二进制码存储数字，而其余的英文字母、标点符号也是以二进制码存储。例如 'A'（字符 A）就可以 0100 0001B（41H，1 byte）来存储。计算机常用的编码方式有两种：ASCII 码和 EBCDIC 码。

1.7.1 ASCII

美国标准信息交换码（American Standard Code for Information Interchange 简称 ASCII）几乎可使用在所有微型计算机（包括 IBM-PC 和 IBM PS/2 机器）上。ASCII 码实际上只使用了 7 位，所以有 128 个 ASCII 字符。余下的第 8 位，即最高位，常用作为校验位，意即在传送或接收字符时仍旧使用 8 位，而在解释这个字符意义时，会将最高位忽略。在 IBM-PC 上也使用 extend character set 扩展字符集。值（80H～FFH）表示图形符号、希腊字符、线条、符号、科学符号。

表 1-6

字　符	Binary	Hex
'A'	01000001	41H
'B'	01000010	42H
'C'	01000011	43H
'1'	00110001	31H
'2'	00110010	32H
'3'	00110011	33H

1.7.2 EBCDIC

EBCDIC（Extended Binary Coded Decimal Interchange Code）码使用 8 位表示一个字符，最多可以表示 $2^8=256$ 个不同的字符。大部分 IBM 计算机均使用这种编码方式来存储文字、数字或符号。连同校验位算在里面，每一字符用 9 位表示。大多数的计算机使用偶校验。

所谓偶校验是将一个字符中所有表示字符内容的位串“1”的数目加起来，若为偶数个，则设校验位为 0，否则为 1。使其保持偶数个为 1，奇校验则须保持奇数个为 1。

在写汇编语言时，所有变量和指令会比高级语言的限制少，但也相对地提高了危险性，许多细节操作都须程序员自己做。

第 2 章　硬件与软件概念

这章解释用汇编设计程序的一般概念。第 1 节复习一些可用的处理器和操作系统以及它们如何协同工作。第 2 节起将开始 CPU 的寄存器组织。如果你不知有那些寄存器可以使用，那大概也不必写汇编语言程序了。就如同不懂九九乘法表要学算术的情况是一样的。第 3、4 节描述系统软件与内存之间的关系，这也是相当重要的基础概念。最后一节摘要描述 MASM 所提供的许多程序设计过程中会用到的工具。

2.1　与汇编语言关系密切的计算机硬件

1. CPU (Central Processing Unit)　这是每台计算机必备的部件，常以其代号代表此计算机名称。常见的有 XT、AT、80286、80386、80486、Pentium。越后面的功能越强大，速度越快。例如 8086（有 16 位的数据总线）就比 8088（只有 8 位的数据总线）快。

16 位数据总线允许你在 8086 处理器使用 EVEN 和 ALIGN 而拥有 WORD 边界的数据，因此可增加数据处理的效率。在 80486 以上的 CPU 大多内置数学浮点运算器 FPU (Floating Point Unit)，这对于 CAD/CAM、3D 等绘图软件的执行速度有很大的帮助。而随着 WINDOWS 与多媒体的流行，大量图形处理对 CPU 速度的要求也越来越高，可见选择适合自己的 CPU 也是很重要的。

2. Memory（RAM、ROM）　RAM (Random Access Memory)，在主机板常看到有 256KB、512KB　SRAM Memory 还有 RAM 插槽上有扩充 RAM，就是我们所常讲的 1MB、4MB、8MB、16MB 的 RAM。ROM (Read Only Memory)，在主机板上常见有 ROM BIOS（如 AMI、AWARD 等）。

3. Serial Port（串行口）　Parallel Port（并行口）　一个 Serial Port（串行口），也称为 Asynchronous（异步）Port。通常在 IBM-PC 做异步传输时，我们使用标准 RS-232 接口。DOS 提供了两个串行口称为 COM1、COM2。它可以连接 MOUSE 或 MODEM。Parallel Port 并行口可同时传送 8 位数据，DOS 提供 3 个 Parallel Port 称为 LPT1、LPT2、LPT3，通常用来连接打印机。

2.2　CPU 寄存器

在 CPU 内部有一些寄存器 (Register)，可用来储存临时数据、内存地址及执行的指令。由于它位于 CPU 内部，可直接接受 CPU 控制，不必通过内存接口按照地址来传送数据，所以具有高速的处理能力。汇编语言能直接存取寄存器，这无形中提高了速度。8086 家族处理器有相同的基本 16 位寄存器集合。

每个处理器也可将某些寄存器视为两个分离的 8 位寄存器。80386/486 有扩展的 32 位寄存器。为维持与其它寄存器的兼容性，80386/486 可以存取寄存器如 16 位或 8 位值。图 2-1 说

明在所有以 8086 为基础的处理器所共有的寄存器，每个寄存器有它自己特殊的使用限制。

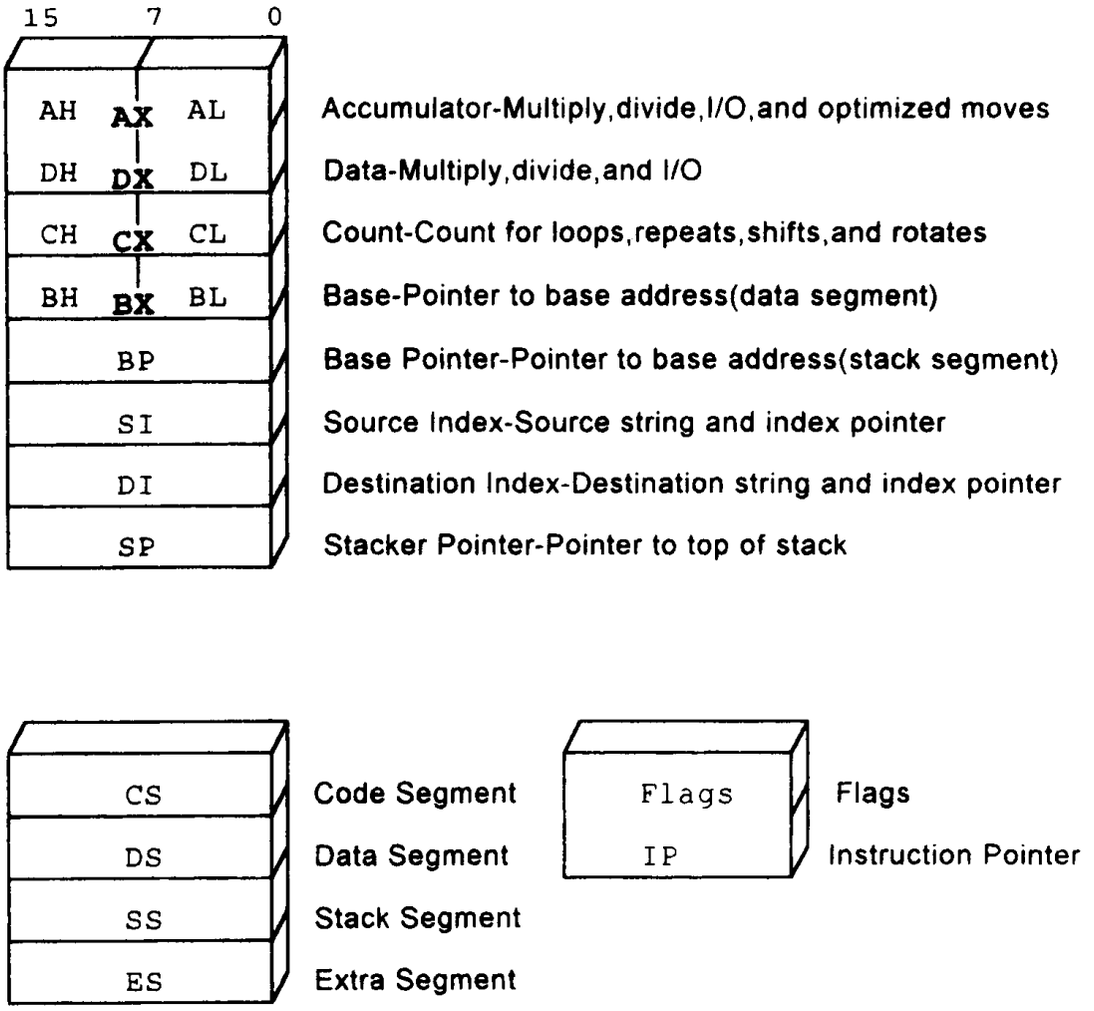


图 2-1　8088、80286 处理器的寄存器

2.2.1　通用寄存器（General-Purpose Register）、**段寄存器**（Segment Register）**和其它寄存器**

80386/486 处理器使用相同 8 位和 16 位寄存器（其余 8086 家族使用）。所有寄存器（除了段寄存器总是占据 16 位）都可扩展成 32 位。这些扩展扩充的寄存器的名字开头都使用字母“E”。例如 AX 的 32 位为 EAX。80386/486 还有两个额外的段寄存器，FS 和 GS。图 2-2 说明 80386/486 的扩展寄存器。

2.2.2　段寄存器

有 4 个主要段寄存器（16 位），CS、DS、SS、ES。（80386/486 增加了 FS 和 GS）。我们在设计程序时，可将 FS/GS 视为数据寄存器来存储较少使用到的数据。在执行时，所有地址都是与以上的段寄存器相关联。这些寄存器，他们的段和他们的目的如下：

寄存器和段	目　　的
CS（Code Segment）	存储代码段起始地址。包含处理器指令和它的立即运算符
DS（Data Segment）	存储数据段起始地址。正常是包含由程序配置的数据

此为试读，需要完整PDF请访问：www.ertongbook.com