朱玉龙　任文岚　朱彤　著

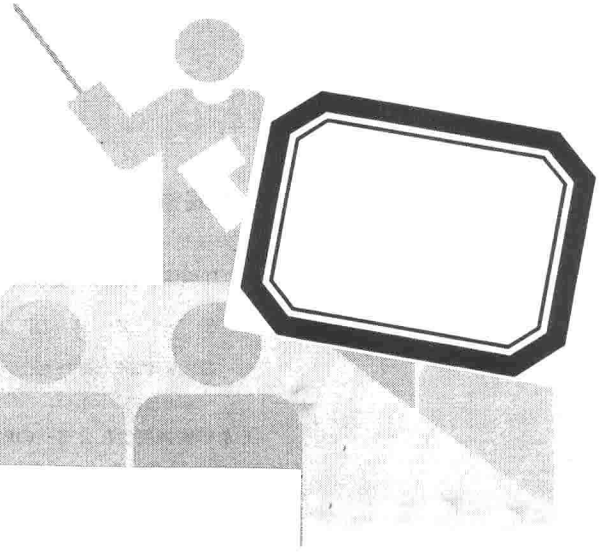
汇编语言轻松编程

- 简便的汇编开发界面
- 完善的输入输出工具
- 75个典型的完整实例
- 方便地调用C标准函数
- 全屏源级与机器级排错

科学出版社

汇编语言轻松编程

朱玉龙　任文岚　朱彤 著



- 简便的汇编开发界面
- 完善的输入输出工具
- 75个典型的完整实例
- 方便地调用C标准函数
- 全屏源级与机器级

科学出版社

内 容 简 介

本书的目的是使汇编语言的编程和学习变得更轻松、更有趣、更有效。为此作者开发了一个短小、强大、方便的汇编语言开发工具 ZASM，其中包括一个集成开发环境和一套输入输出宏。三年的教学实践表明，ZASM 大大加速了汇编语言的学习进程，得到普遍欢迎。

本书包含十章和三个附录。书中注重学生设计实验、验证知识、寻求答案。为培养学生的学习兴趣，所有例题都是完整的、正确的、有趣的，籍以说明只要掌握了编程规律，汇编语言并不比 C 语言更难学。

本书不要求任何前提课程，适于用作本科和专科汇编语言程序设计课程的教材，也可供计算机专业的研究生和软件设计人员参考。

汇编语言轻松编程

朱玉龙 任文岚 朱 彤 著

责任编辑 孙桂荣 史增启

科 学 出 版 社 出 版

北京东黄城根北街 16 号

邮政编码：100717

丹东印刷有限责任公司 印刷

科学出版社发行 各地新华书店经销

*

2001 年 7 月第 一 版 开本：787×1092 1/16

2001 年 7 月第一次印刷 印张：17 3/8

印数：0001-5000 字数：398 000

ISBN 7-03-007969-8/TP·1269

定价：27.80 元(附软盘)

前　言

我们有幸生活在一个信息爆炸的时代，每天都能享受到许多精彩、便利的软件工具，同时我们必须不断地学习。然而要学的东西太多，我们的时间又太少。

时代在发展，计算机专业的课程表也在不断地更新，然而最古老的汇编语言仍然是必修课程，原因有三。第一，它是最基本、最快速的编程语言。对于某些应用，汇编语言可能是唯一选择。第二，没有汇编语言基础，就不能学习诸如操作系统、编译原理等课程，甚至不能真正学好象 C 语言这样的课程。第三，汇编语言是与计算机识别的唯一语言——机器语言同级的语言。汇编语言使学生能够近距离地观察计算机的内部操作，而且汇编语言中蕴涵着必不可少的独特思想和技巧，大凡成功的计算机人士都具有良好的汇编语言修养。

笔者多年从事汇编语言编程，接触过多种机器的汇编语言，感到唯有 PC 机汇编语言最难学，其实 PC 的汇编语言本身并不比其他汇编语言复杂。因此总想写一本与众不同的书，把自己的经验和心得写进去，使得汇编语言的编程和学习变得更轻松。于是有了这本书。在写作本书过程中，笔者有以下考虑：

- 计算机工作者面对一个迅速改变的世界，他必须善于学习。知识重要，获取知识和运用知识的能力更重要。知识会过时，能力却不会。本书讲的是汇编语言，要让学生学会使用汇编语言解决实际问题，但更重要的是通过汇编语言的学习，让学生掌握用计算机去学习计算机知识的方法。通过上机实验、观察、对比、推论，这样学习更轻松，而且取得的知识更准确、更深刻。
- 兴趣是学习的催化剂，没有兴趣的学习是痛苦的。如果我们能在培养学生的专业兴趣方面有所作为，学生将受益非浅。本书在选材和论述方面力求有趣、精彩、引人入胜。
- 程序必须能够对于不同输入做出反应，没有 I/O 的程序算不上是真正的程序，而且不易理解，不易讲解，不易排错。汇编语言不具备有效的 I/O 手段，汇编语言的学习往往变得索然无味，学习效果自然大打折扣。为此笔者编写了汇编语言的 I/O 功能库，功能完善，能够读、写字符、字符串、数值和数组，具有格式控制能力，其用法极为简单，学生很快就能用它写出真正的程序，编程变得轻松、有趣，学习容易见效。另外，这个工具虽小五脏具全，是难得的应用实例。书中随着课程的进展把其中的片段陆续介绍给学生，并鼓励他们去修改和扩展，这对于培养学生动手能力极为有利。
- 开发汇编语言程序需要反复敲入一些键盘命令，大部分上机时间浪费到机械的敲键动作上，麻烦、低效、容易出错。笔者曾努力寻找类似于 Turbo C 的汇编语言集成开发环境，结果总不尽人意。不得已笔者编写了这样一个环境 ZMEN，它覆盖了编辑、汇编、连接、排错、运行、存盘，以及观察各种中间文件等操作，按一下键就执行一条命令，避免了键入错误；而且屏幕上显示着执行的命令，使学生在享受便利的同时熟悉了所有的程序开发命令。
- 在 ZMEN 中有一个汇编语言程序框架，每当学生要新编一个程序时都将自动此框架，作为开发的基础，省时省力避免了一些错误。
- 汇编语言程序的排错是非常艰巨的工作，对于初学者往往意味着焦虑和无奈，提高排错能力应该是我们教学的目标之一。大部分教材给学生提供早期的排错程序 DEBUG，不

方便、效果差；其余的使用微软的 CV，但 CV 是一个中间产品，不甚成功。本书采用 Turbo Debugger（简称 TD）。这是一个全屏幕的排错程序，直观、方便、有效。

- 本书强调能力的培养，突出原理、创意和技巧，不追求全面，因此要“少而精”，只有少才能精，才能深。细心的读者不难发现本书少了一些“冗余”，同时也增加了许多“精深”的内容。
- 学习程序设计之初总要模仿，例题的质量在很大程度上决定了教学的水平。本书提供了 70 多个完整的程序例题，每题都力图说明一种概念或技巧。所有程序都是正确的、易懂的，都有运行实例。读程序也是一种必要的训练，书中的某些例题可以留给学生自己阅读。

本书分为十章和三个附录。第 1 章基础知识，包括数制的转换，微机体系结构和信息在计算机中的表示方法。第 2 章介绍命令级操作，包括上机开发程序使用的命令，特别详细讨论了排错工具 TD 以便能在机器上学习机器指令，另外还描述了笔者编写的编程环境 ZMEN。以下两章讨论汇编语言中的指令——第 3 章讨论符号指令，第 4 章包括汇编伪指令和笔者提供的 I/O 功能库的用法。后面六章详细探讨各种程序设计技术。第 5 章研究结构化程序设计技术在汇编语言中的实现，以及顺序结构和分支结构的编写方法，第 6 到 8 章学习的内容分别是循环结构、过程和宏指令的实现技术。第 9 章介绍在开发大型应用时采用的模块化程序设计技术，并作为例子给出了开发一个实用的 I/O 功能库的全过程，另外还说明了如何在汇编语言程序中调用 C 语言的标准函数。第 10 章开始于 BIOS 和 DOS 功能调用的简介，而后详细地介绍了文件代号式的文件系统及其使用方法。附录 A 补充了 EDIT 和 TD 的功能键定义卡片，附录 B 和附录 C 分别给出微机指令系统表和伪指令表，相信会给读者带来方便。

第 5～8 章中每章的最后一节给出了一些典型而有趣的参考例题，读者可以根据时间情况选读其中的一部分。本书配有一张软盘，包含了书中全部例题和笔者开发的编程工具 ZASM。

在本书写作期间得到了系主任吉根林副教授的鼓励；蒲宝明研究员、张正兰教授和张明教授审阅了本书；崔海源副教授、吕升旭副教授和张淮中讲师以本书初稿作为教材讲授了汇编语言程序设计课程，并提出很多宝贵的建议。对于他们的帮助，在此致以衷心的感谢。

本书的写作历时三年，为消除错竭尽全力，但由于水平有限，书中一定还有许多错误和不妥之处，衷心地期待读者批评指正。

朱玉龙
南京师范大学计算机系
2001 年 1 月

目　　录

注意：标有*的小节可以留作阅读练习。

第一章 基础知识

本章介绍预备知识，包括数制和数制之间的转换，微型计算机系统的体系结构，以及各种信息在计算机中的表示方法。

1.1 数制及数制之间的转换

人们计数和计算使用十进制，计算机内部使用二进制，人们谈论计算机内部的数用十六进制。本节讨论怎样用这三种进制表示整数，以及在这三种进制之间如何进行转换。

1.1.1 十进制、二进制与十六进制

所谓“数”是一个抽象的概念。自然界里本没有1，也没有2，有的只是一个苹果，两个梨。经过很长很长时间，人们从许许多多客观事物中得到了1和2这样的数的概念。为了记忆和交流的目的，他们需要把数表示出来，最初使用的可能是石块和绳结。

后来人们得到了多种数的表示方法，最常用的就是我们所熟悉的**十进制**。十进制的基数是10，十进制的特点是，十个数码，逢十进一。具体地说，每个十进制数都由十个数码0、1、2、...、9构成，而且这些数码在不同的位置上有不同的位权值，邻位之间逢十进一。个位（第0位）的位权值是$10^0=1$，十位（第1位）的位权值是$10^1=10$，依此类推。如果位号从最低位开始向右排起——把个位叫做第0位，十位叫做第1位的话，那么容易写出各位的位权值：

位号	6	5	4	3	2	1	0
十进制数=	1	1	1	1	1	1	1
位权值	10^6	10^5	10^4	10^3	10^2	10^1	10^0

一个十进制数的值等于它的所有位上的数码与对应位权值之积的和。作为公式，一个具有n位数字的十进制数同其值的关系是：

$$d_{n-1}d_{n-2}\ldots d_1 d_0 = d_{n-1}\times 10^{n-1} + d_{n-2}\times 10^{n-2}\ldots + d_1\times 10^1 + d_0\times 10^0$$

例如，十进制数：

$$123456=1\times 10^5+2\times 10^4+3\times 10^3+4\times 10^2+5\times 10^1+6\times 10^0$$

我们使用十进制感到很方便，大概是因为我们有十个手指。自从计算机发明以来，二进制大行其道。**二进制**的基数是2，只有两个数码：0和1；运算特别简单，加法逢二进一：0+0=0，0+1=1，1+0=0，1+1=10。这里的10是二进制的10，等于十进制的2。为避免混淆，我们在十进制数的后面标示以大写或小写D字母，而在二进制数的后面标示以B字母。一个二进制数第i位的位权值就是2^i，因此有以下图示：

位号	10	9	8	7	6	5	4	3	2	1	0
二进制数	1	1	1	1	1	1	1	1	1	1	1
位权值	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
=	1024	512	256	128	64	32	16	8	4	2	1

例如，二进制数：

10110111B=128×1+64×0+32×1+16×1+8×0+4×1+2×1+1×1＝183D

尽管二进制运算简单，但是它有个很大的缺点，就是写起来太长，比如上面的数用十进制表示占 3 位，用二进制占 8 位。为了写得短一些，我们经常使用**十六进制**，其实十六进制可以看作是二进制的缩写。顾名思义，十六进制要有十六个数码，因为我们常用的数码只有十个（即从 0 到 9），不够用，还需补充 6 个。我们补充的是大写或小写的字母 A～F。请看表 1-1。

根据此表，我们可以把一个二进制数很容易地缩写成十六进制：从最低位开始每四位分做一组，若最后一组不足四位则用 0 补足，然后每组用对应的十六进制数码代替之。注意，十六进制数的标志字符是 H，另外得到的十六进制数的最高位如果是字母的话，则在前面补一个 0 以表明它是一个数，不是通常的名字。例如前面的二进制数可以表成：

表 1-1 十六进制数码的定义

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

1011 0111B = 0B7H = 183D

反之，一个十六进制数可以同样容易地扩展成二进制数：将十六进制数的各位数码依次换成表中对应的 4 位二进制数。如 0B7H 的最高位是 B，对应的二进制数是 1011，而次高位 7 对应的二进制数是 0111，把它们合起来即为 10110111B。这就是**二——十六进制之间的转换**。

1.1.2 十—十六进制之间的转换

现在我们有了十进制、二进制和十六进制这三种进制，那么在它们之间有六种转换，如图所示。其中二与十六进制之间的转换最为简单，前面已经讨论过了。而二与十进制之间的转换虽然计算简单，但是计算过程较长，比较麻烦。本节只讨论十与十六进制之间的转换。二与十进制之间的转换，可以用二与十六进制之间的转换以及十与十六进制之间的转换来实现。

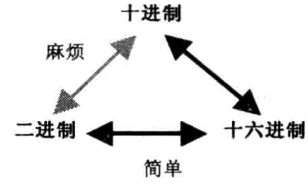


图 1-1 三种进制的转换

十→十六进制的转换

有两种实现方法：除基法和 除权法。**除基法** 是用十六进制的基数 16 去除被转换的十进制数，将所得余数写成十六进制数码作为所求的十六进制数的最低位，而后将除得的商再除以 16，并把所得余数写成十六进制数码作为次低位，依次类推，直到商小于 16 时为止，这时此商就是最高位。

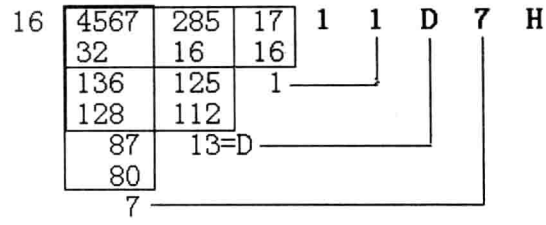


图 1-2 用除基法把十进制数转换成十六进制数

例如，有十进制数 4567D，则可以采取图 1-2 中的纵式的计算过程：

位号	4	3	2	1	0
十六进制数	1	1	1	1	1
位权值	16^4	16^3	16^2	16^1	16^0
=	65536	4096	256	16	1

首先 4567 除以 16，余为 7——最低位，商 285 再除以 16，余为 13——表成十六进制数码 D，作为次低位，其商 17 除以 16，余为 1，商也为 1，计算结束，得到十六进制数 11D7H 。

第二种方法是**除权法**。 为使用除权法，需要记住十六进制的各个位权值 ，经常使用的只有第 0 位到第 3 位的位权值。请看下面所列的位权值：

除权法的步骤是：首先找出不大于被转换十进制数的最大位权值，用此数去除被转换数，把得到的商写成十六进制数码，这就是转换后的十六进制数的最高非 0 位，然后再将余数除以下一个较小的位权值，从而得到转换后的十六进制数的下一位，照此办理直至余数小于 16 时为止，这个余数就是转换后的十六进制数的最低位。

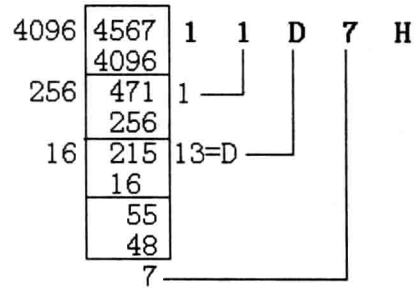


图 1-3 用除权法把十进制数转换成十六进制数

请看图 1-3 的运算过程：首先把 4567 除以 4096，商为 1——十六进制数的最高位数字，将余数 471 除以 256，商仍是 1——次高位数字，再将余数 215 除以 16，商 13 即十六进制数码 D——第 1 位，余为 7，小于 16，因此停止相除，7 是最低位。

十六→十进制的转换

与十→十六进制的转换类似，十六→十进制也有两种转换方法：乘基法和乘权法。**乘权法**依据的是公式：

$$h_{n-1}h_{n-2}\ldots\ h_1\ h_0 = h_{n-1}\times 16^{n-1} + h_{n-2}\times 16^{n-2}\ldots\ + h_1\times 16^1 + h_0\times 16^0$$

这里所有的十六进制数码和各个位权值都要换成等值的十进制表示，并采取十进制计算，计算结果自然也是一个十进制数了。例如，

11D7H=1×4096+1×256+13×16+7×1=4567D

而**乘基法**根据的公式是：

$h_{n-1}h_{n-2}\ldots h_1 h_0=((\ldots(h_{n-1}\times16+h_{n-2})\times16)+\ldots+h_1)\times16+h_0$

这种方法的好处就是乘数总是 16。例如，

11D7H=((1×16+1)×16+13)×16+7

=(17×16+13)×16+7

=285×16+7

=4560+7 =4567D

在用汇编语言编写程序时经常要用笔和纸进行三种数制之间的转换，因此我们需要掌握一套转换方法就可以。这套方法应该包括简单的二与十六之间的转换、一种十→十六转换法和一种十六→十转换法。

1.2 微机的体系结构

硬件是软件的基础，软件是硬件的延伸；硬件是软件的舞台，软件是舞台上的角色。计算机程序只在计算机上才有生命。因此学习计算机语言，特别是汇编语言，必须对于计算机硬件有所了解。

计算机硬件极为复杂，只为学习微机的汇编语言，我们不可能也不需要了解微机的全部细枝末节。我们需要了解的正是微机中程序可以感知和改变的的部分，或者说是微机的体系结构，即微机的功能描述，包括内存寻址方式、指令系统、中断功能等。

一台微机通常包括中央处理机、存储器和输入输出（I/O）子系统三部分，相互之间用系统总线连接起来。

中央处理机（CPU）要控制整个机器，执行全部指令。

存储器 通常也叫做**内存**，用于存放 CPU 执行的全部指令，以及执行指令时所需要的数据和产生的中间结果。它离 CPU 很“近”，CPU 可以随取随用其中的指令和数据。它是一种易失性存储器，就是说，电源一关，它所存放的内容就全部丢失。

I/O 子系统 包括 I/O 设备和大容量存储器。I/O 设备是计算机同外部世界联系的媒介，能够给计算机提供要执行的程序和必要的数据，还要把计算机的计算结果告诉人们，如显示器、键盘、鼠标、打印机等。大容量存储器，也叫做外部存储器，简称“外存”，其特点是容量大，而且可以长期保存数据，即使关了电源，其内容也不破坏。

本书介绍的微机是用 Intel 的微处理器作为 CPU 的。在过去的 20 年里，Intel 先后推出了 8088/80188、8086/80186、80286、80386、80486、Pentium（或称为 80586）、PentiumPro（或称为 80686）以及 Pentium III。它们在运算速度、内存容量、寄存器和总线的宽度以及指令的条数方面各不相同。但是它们都是向下兼容的，就是在低档机器上编写的程序不加改变就能在高档机器上运行。鉴于篇幅限制，本书只讨论 8088 和 8086 处理器（其差别甚微）的程序设计。

1.2.1 8086 处理器

如图 1-4 所示，处理器分成两个逻辑部件：执行单元（EU）和总线接口单元（BIU）。EU 的作用就是执行指令，而 BIU 的任务是给 EU 提供指令和数据。EU 又可以分为算术逻辑部件 ALU、控制部件 CU 和一些寄存器。

BIU 最重要的功能是管理总线控制器、段寄存器和指令队列。在总线控制器的控制之下，总线把数据传送给执行单元、内存和输入输出（I/O）设备。段寄存器用来形成内存的

物理地址。

BIU 的另一个作用是取指令。因为程序所执行的指令放在内存之中，BIU 必须从内存中取来指令，并把它们放在一个指令队列中。有了指令队列，BIU 才能预先查找并取来指令，于是指令队列中总有一些准备执行的指令。EU 和 BIU 并行工作，只是 BIU 总要领先 EU 一步。当 EU 需要内存中的数据或者需要访问 I/O 设备时，它就通知 BIU。此外，EU 还要 BIU 给它提供已存放在指令队列中的指令。队列顶部的指令就是正在执行的指令。当 EU 执行一条指令时，BIU 就从内存中取出下一条指令。取指令与执行指令的重叠，提高了处理的速度。

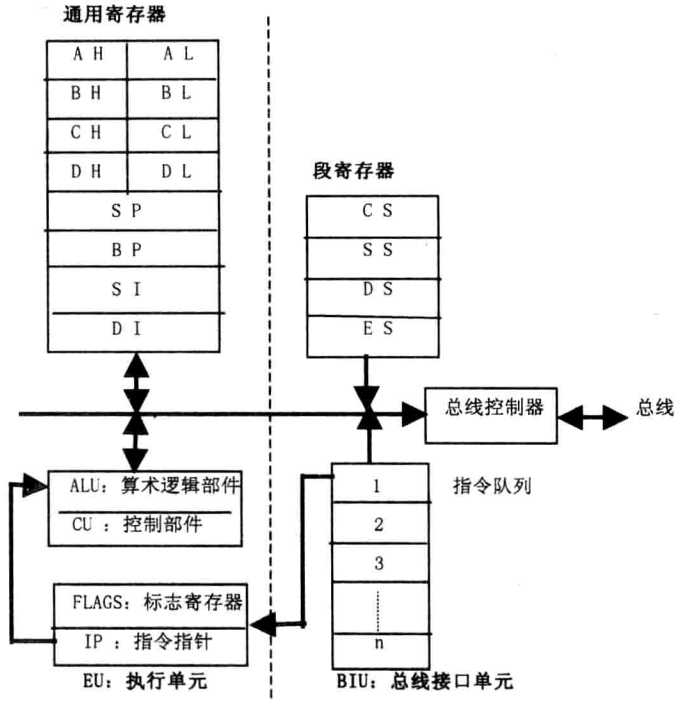


图 1-4 8086 处理器的构成

在图 1-4 中可以看到对于程序员极为重要的 14 个寄存器，因为它们几乎是程序能够看到的全部硬件，所以有时被叫做程序设计模型。按用途分，这些寄存器可以分成两类：通用寄存器和专用寄存器，如图 1-5 所示。

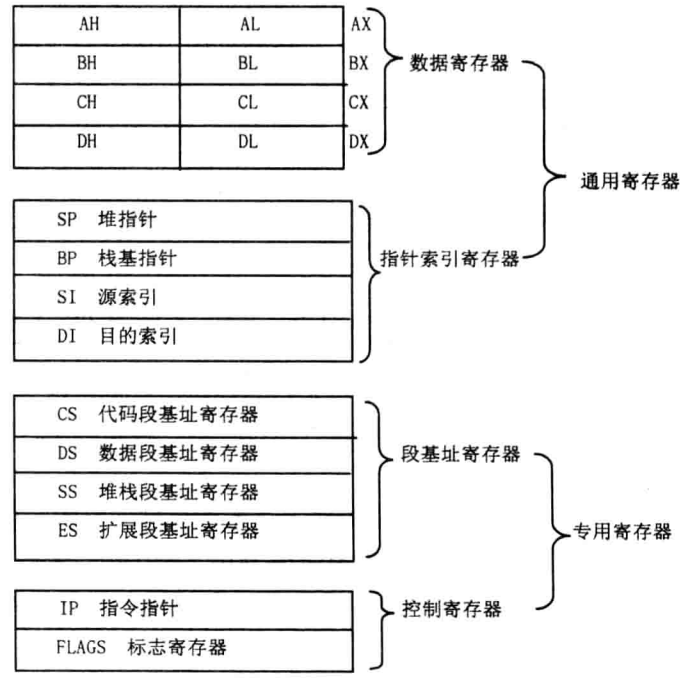


图 1-5 8086/8088 的寄存器

通用寄存器共有 8 个，长度都是字，即 16 位。它们的共同特点是可以作为操作数执行算术、逻辑、移位等运算，所以统称为通用寄存器。其中的 SP、BP、SI、DI 主要用于构成偏移，也叫做指针索引寄存器。具体地说，SP 总指向堆栈的栈顶，所以叫做堆栈指针（Stack Pinter）；BP 为过程（子程序）访问存放在堆栈中的参数和变量提供基础，不妨叫做栈基指针（Base Pointer）。SI 和 DI 在串指令中分别指向源串和目的串，从而有源索引（Source Index）和目的索引（Destination Index）之称。

表 1-2 各数据寄存器的特殊功能

数据寄存器	特殊功能
AX，AL（累加器）	在乘法指令中存放被乘数和乘积的低阶部分 在除法指令中用于存放被除数的低阶部分和商 在 I/O 指令中用于存放与 I/O 端口交换的数据
AH	在 INT 指令中常用于存放功能号
BX（基址索引）	常用做基址寄存器，参与构成偏移
CX（计数器）	在循环指令和串指令中存放循环执行的次数
CL	在移位指令中存放移位的位数
DX	在乘法指令中存放乘积的高阶部分 在除法指令中存放被除数的高阶部分和余数 在 I/O 指令中存放 I/O 端口的编号 在系统中断服务中常存放相关偏移

另外的四个寄存器 AX～DX 更多地用于运算，所以把它们合称为数据寄存器。它们中的每一个都是字寄存器，并且还可以当作两个字节寄存器用于字节运算。例如，AX 的高、低字节分别叫做叫做 AH 和 AL，在进行字节运算时它们是相互独立的字节寄存器。需要指出的是，在进行通常的计算方面，4 个字寄存器彼此之间并无区别，8 个字节寄存器亦然。但是它们各自又有一些独特功能，表 1 -2 概括了各数据寄存器的特殊功能。另外还要注意，BX 虽属数据寄存器，但也常参与构成偏移。

四个段寄存器 CS、DS、SS 和 ES，位于 BIU 中，分别装有代码段、数据段、堆栈段和扩展段的段基址。在 8086 上，内存单元的逻辑地址由一个 16 位的段基址和一个 16 位的偏移所组成，偏移由指令给出，段基址则由段寄存器提供。

两个控制寄存器是 IP 和 FLAGS。IP 是指令指针（Instruction Pointer），总装有下一条要执行的指令的偏移，BIU 将 CS 中的段基址和 IP 中的偏移合起来，构成物理地址，按此地址从内存中取得下一条指令，随即把此指令的长度加到 IP 上使之指向下一条指令。

FLAGS 是标志寄存器，占一个字，只用了 9 位，这些位的名称和位置如下图所示：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

标志位分成两种：一种是状态标志，由机器设置供程序检查，能够描写机器的当前状态和上一条运算型指令的执行情况；另一种是控制标志，由程序设置以控制机器的操作。各标志位的意义说明如下：

1. ZF（Zero）零标志,指明上一条运算型（算术、逻辑、移位等）指令的运算结果是否为零。如果 ZF=1，表示结果为 0；如 ZF=0，表示结果不为 0。
2. SF（Sign）符号标志，取自上一条运算型指令的运算结果的符号位（最高位）。如为 0，表示结果为正；为 1，结果为负。
3. OF（Overflow）溢出标志，当把运算数视为有符号数时，指明运算结果是否超出了有符号数的表示范围。如为 0，表示结果正确；如为 1，表示超出范围，结果错误。
4. CF（Carry）进位标志，当把运算数视为无符号数时，指明运算结果是否超出了无符

号数的表示范围。如为 0，表示结果正确；如为 1，表示超出范围，结果错误。对于加法运算，它实际上是最高位向上的进位，因此叫做进位标志。

5. PF（Parity）奇偶标志，这是早期 Intel 微处理器为数据通信的正确性检验（奇偶检验）所提供的，现已过时。它记录了运算结果的低字节中取 1 的位数的奇偶性，PF=1 表示有偶数个 1，PF=0 表示有奇数个 1。
6. AF（Auxiliary Carry）辅助进位标志，只用于十进制调整。对于加法运算，它是第 3 位到第 4 位的进位。本书不涉及。
7. DF（Direction）方向标志，为串操作指令指明地址改变的方向，DF=0 表示地址增加，DF=1 表示地址减少。指令 STD 和 CLD 分别用于将 DF 置 1 和清 0。
8. IF（Interrupt）中断标志，指明是否需要处理外部中断。IF=1 表示允许外部中断发生，IF=0 表示要忽略外部中断。指令 STI 和 CLI 分别用于将 IF 置 1 和清 0。
9. TF（Trap）陷井标志，如果此标志位为 1，则处理机进入单步执行方式。各种排错工具，如 Debugger、CodeView 和 Turbo Debugger，都使用这个标志位，所以你才能一次执行一条指令、随时查看各个内存单元，以及设置断点、观察点。

不难看出，标志 DF、IF 和 TF 是控制标志，其余 6 个都是状态标志。

1.2.2 内存

内部存储器，简称内存。PC 机有两种内存：随机存取内存（RAM）和只读内存（ROM）。内存中最小的存储单元是字节。内存中的全部字节从 0 开始依次编号，所以每一个字节都有一个唯一的物理地址。图 1-6 给出了 8086 微机的内存分配情况。它的 1M 字节内存中，前 640K 是基本 RAM，其中的绝大部分可供用户程序使用。

ROM 的意思是只读存储器（Read Only Memory），是由一种特殊的存储芯片组成的，只能读不能写。因为一些指令和数据被“烧结”在里面，所以没法改变。ROM 中有一个基本输入输出系统（BIOS），从地址 768K 开始，负责管理 I/O 设备，诸如硬盘控制器。从 960K 开始的 ROM 用于实现计算机的基本功能，如上电自测试、图形点阵、磁盘自举。当你接通电源时，ROM 中的程序要完成各种各样的检查，还要从磁盘上取来特定的系统数据并装入内存。

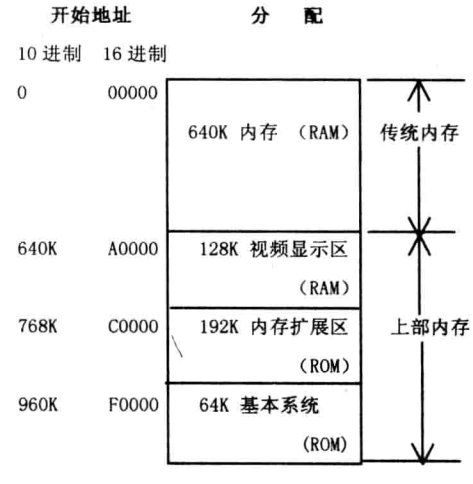


图 1-6 内存分配图

RAM 程序员关心的主要是 RAM，意为随机访问存储器（Random Access Memory），其实叫做可读写存储器也许更准确。它可以反复记录数据，所以可以用作临时存储，也可以存放执行的程序。以后我们将使用“内存”一词来称呼 RAM。

因为在切断电源时 RAM 的内容会统统消失，所以我们需要有单独的外部存储器——磁

盘来保存程序和数据。当接通电源时，ROM 中的自举过程会把操作系统的一部分从磁盘装入到 RAM。随后我们就可以要求操作系统完成某种动作，诸如把某个程序从盘装入到 RAM。程序只能在 RAM 中执行，执行时可能要把计算的结果输出到显示器、打印机或者磁盘上。这个程序执行完了，我们可以要求系统再装入另一个程序，新装入的程序将覆盖原来的程序。

内存最小的存储单元——**字节**（Byte），由 8 个二进位组成。两个相邻的字节可以构成一个字（Word），由 16 位组成。字节和字都是 8086 的存储单元，可以用一条指令把一个字节或一个字写入内存，也可以用一条指令从内存中读出一个字节或一个字。进而，字节和字也是 8086 的计算单位，几乎所有指令都是以字节或字作为操作对象的，就是说，CPU 能执行 8 位或 16 位的运算。

8086 有 20 条地址线，能够表达的最大地址是 $2^{20}-1$=0FFFFH，所以它的直接寻址能力就是 1MB，（读作 1 兆字节，等于 1 百万字节）。只要给出一个 20 位的地址，就可以从这 1MB 中唯一地选取一个字节来。因此我们说 8086 内存的**物理地址**是 20 位的。

然而，由于最大的存储单元和运算单位是 16 位的字，所以只能用字表示内存的地址。一个字能表示的范围是 0 到 0FFFFH，仅达到 64K，无法访问其余的 15/16 内存。那么怎样表示地址才能使之达到整个 1M 的内存呢？可能有很多的选择，这里采取了“段基址:偏移”表示法。我们用两个字来表示一个内存地址，第一个字叫**段基址**，第二个字叫**偏移**，它们合起来构成了一个**逻辑地址**。BIU 在接到了一个逻辑地址的时候，先要把这个由两字构成的逻辑地址映射成 20 位的物理地址，而后用此物理地址选取内存中的指定单元。这个地址映射是这样实现的：段基址向左移动 4 位（相当于乘 16），再加上偏移，得到 20 位的物理地址。这个过程可用图 1-7 表示，也可写成公式：

段基址×16d＋偏移＝物理地址

例如，逻辑地址 0100H:02A0H 对应的物理地址是：

0100H×16d＋02A0H

=01000H＋02A0H=012A0H

显然，内存中的每一个字节都只有一个物理地址，但是却可以有多个逻辑地址。比如说逻辑地址 0101H:0290H 对应的物理地址也是 012A0H。

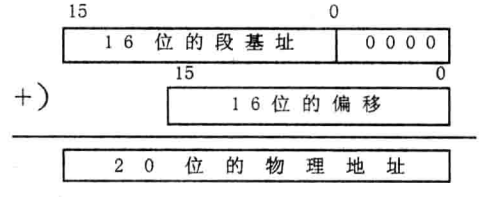


图 1-7 从逻辑地址到物理地址的映射

通过上面的地址映射过程，我们得知，任何段的开始物理地址的末四位只能是 0。我们把从末四位为 0 的物理地址开始的、由 16 个字节组成的区域叫做**节**（Paragragh）

段基址必须存放在段寄存器中才能起作用，8086 有四个段寄存器：CS、DS、SS 和 ES。因此一个程序可以访问许多个段，但是在任意时刻只能访问 4 个段的内容，在需要访问其他段的时候需要将那个段的段基址放在适当的段寄存器中。

四个段寄存器承担的责任是不同的。CS 装的是代码段的段基址，代码段包含正在执行