# 第 1 章　基础知识

## 1.1　计算机系统概述

计算机系统包括硬件和软件两部分。硬件包括各种功能部件电路、外部设备和机柜等硬设备。软件则是为了运行、管理和维护计算机而编制的各种程序的总和。

### 1.1.1　硬件

典型的计算机的基本结构框图如图 1－1 所示。

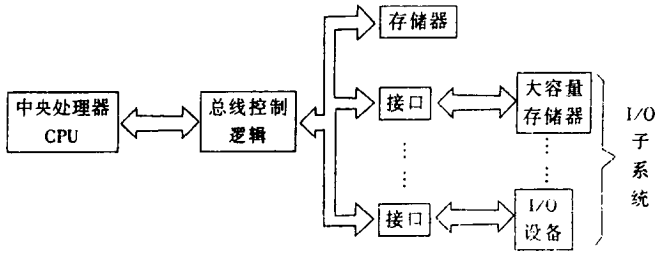



图 1－1　计算机结构框图

其中包括中央处理器 CPU(Central Processing Unit)、存储器(Memory)和输入／输出(Input/Output)子系统三个主要组成部分，它们三者由系统总线连接在一起。

存储器是计算机的记忆部件，人们编写的程序（由指令序列组成）就存放在里面。它也可以存放程序中所用的数据（原始数据和结果数据）、信息及中间结果。一般称这种在机器内部的存储器为内存。

中央处理器包括运算器和控制器、内部可编程寄存器组。运算器执行所有的算术和逻辑运算指令。控制器则负责全机的控制操作，它负责把指令逐条从存储器中取出，经译码分析后向全机发出取数、执行和存数等控制命令，以保证正确完成程序所要求的功能。 I/O 子系统(输入／输出)一般包括 I/O 设备。I/O 设备是指与计算机的外部世界通信用的输入／输出设备.如显示终端、键盘、打印机、磁盘、磁带和光盘等多种类型的外部设备。由于内存容量有限，计算机用外部存储器（磁盘、磁带和光盘）作为内存的后援设备，它的容量比内存大得多，但存取信息的速度比内存慢得多。除了必要的系统程序（如 DOS 的引导程序及直接和输入／输出设备进行数据交换的程序）是存放在内存中以外，一般程序（包括数据）是存放在外部存储器中

的。只有当运行时，才把它们从外部存储器送到内存的某个区域，由中央处理器 CPU 控制执行。

系统总线把 CPU、存储器和 I/O 设备连接起来，用来传送各部分之间的信息。系统总线包括数据总线、地址总线和控制总线，简称三总线。数据总线传送数据（包括指令代码、原始数据、中间数据和结果数据），地址总线上的信息（即地址）指出数据的来源和目的地，控制总线传送 CPU 对存储器或 I/O 设备的控制命令和 I/O 设备对 CPU 的请求信号。系统总线的工作由总线控制逻辑（在 CPU 的控制器内）负责指挥。

## 1.1.2 软件

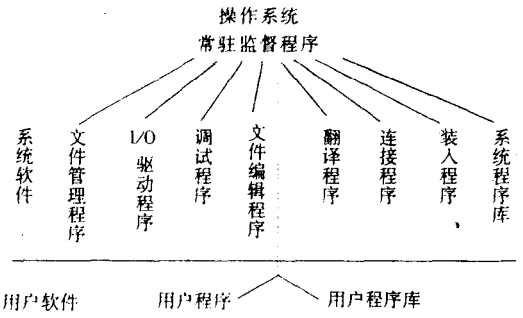



图 1-2 计算机软件层次图

计算机软件是计算机系统的重要组成部分，它可分为系统软件和用户软件两大类。系统软件是由计算机厂商提供给用户的一组程序，这些程序是用户使用机器时为产生、准备和执行用户程序所必需的。用户软件则是用户自行编制的各种程序。图 1-2 表示了计算机软件的层次。下面我们简要地介绍系统软件的组成

系统软件的核心称为操作系统（Operating System）。操作系统是系统程序的集合，它的主要作用是对系统的软硬件资源进行合理的管理，为用户创造方便、有效和可靠的计算机工作环境。

操作系统的主要部分是常驻监督程序（Monitor），只要一开机它就开始运行，它可以接受用户命令，并使操作系统执行相应的动作

I/O 驱动程序（I/O driver）用来对 I/O 设备进行控制和管理。当系统程序或用户程序需要使用 I/O 设备，只要发出命令，执行 I/O 驱动程序，完成 CPU 和 I/O 设备之间的信息传送。

文件管理系统（File Management）：用来处理存放在外存储器中的大量信息，它可以和外存储器的设备驱动程序相连接，对存放在其中的信息以文件（File）形式进行存取，复制及其它管理操作。

文本编辑程序（Text Editor）：文本是指由字母、数字和符号等组成的一组信息，它可以是一个用汇编语言或高级语言编写的程序，也可以是一组数据或一份报告。文本编辑程序用来建立、输入或修改文本，并使它存入内部存储器或外部存储器中（磁盘）。例如，PC 机提供的编辑程序 EDIT 可用来建立源文件、修改文本，有插入、删除、编辑和显示行等功能。中西文字处理程序 WPS、WORD 等可提供屏幕编辑功能，并能提供各种功能及命令菜单，使文本的建立和修改更加方便。

翻译程序（Translator）：它将把用户用汇编语言或高级语言编写的程序（称为源程序）翻译成机器语言程序（称为目标程序）。汇编语言的翻译程序称为汇编程序（Assembler）。高级语言的翻译程序有两种形式：一种是将高级语言的源程序一边进行解释，一边执行，这种翻译程序称为解释程序（Interpreter），如 BASIC 经常采用这种形式。另一种是先把高级语言编写的源程

序翻译成机器语言程序，然后再在机器上执行，这种翻译程序称为编译程序（Compiler），多数高级语言如 C、PASCAL 等都采用这种形式。

连接程序（Linker）：用来把要执行的程序与库文件（子程序库中的子程序）或其它已翻译过的子程序（能完成一种独立功能的程序模块）连接在一起，形成机器能执行的程序。

装入程序（Loader）：用来把程序从外存储器传送到内存，以便机器执行。例如，计算机开机后需要立即启动装入程序把常驻监督程序装入内存，使机器运转起来。又如，用户程序经翻译和连接后，由连接程序直接调用装入程序，把可执行的用户程序装入内存以便执行。

调试程序（Debug）：是系统提供给用户的能监督和控制用户程序的一种工具程序，它可以装入、修改、显示或逐条或连续执行一个程序。在 PC 机上，简单的汇编语言程序可以通过调试程序来建立、修改和执行。

系统程序库（System Library）和用户程序库（User Library）：各种标准程序、子程序及一些文件的集合称为程序库，它可以被系统程序或用户程序调用。操作系统还允许用户建立程序库，以提高不同类型用户的工作效率。

## 1.2 中央处理单元——微处理器 8086/8088

### 1.2.1 微处理器 8086/8088 的组成

Intel 8086/8088 是两种第三代微处理器。在汇编语言一级，它们与 8080/8085 微处理器是兼容的。它具有 20 条地址总线，直接寻址内部存储器能达到 1MB（1 兆个字节存储单元）。8088 具有 8 位数据总线可与内存或输入／输出设备交换数据，而 8086 则有 16 位数据总线。其它方面两个微处理器都是相同的，为其中一个 CPU 编写的软件，可以不加修改地在另一个 CPU 上执行。

CPU 的任务是执行存放在内存中的指令序列。它由运算器、控制器和内部可编程寄存器组所组成。Intel 8086/8088 微处理器的组成示意图如图 1－3 所示。

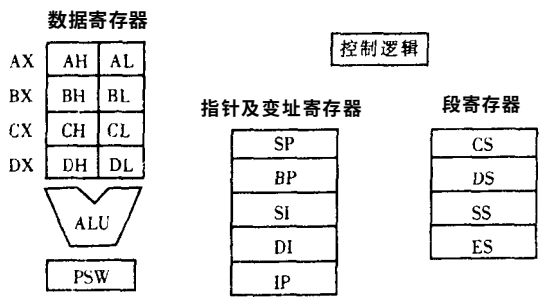



图 1－3　8086/8088CPU 组成

从图中可见 CPU 由三部分组成：

①算术逻辑部件 ALU(Arithmetic Logic Unit)用来进行算术和逻辑运算。

② 控制逻辑负责对全机的控制工作，包括从存储器取出指令，对指令进行译码分析，从存储器取得操作数，发出执行该指令的所有命令，把结果存入存储器，以及对总线及 I/O 传送控制等。

③工作寄存器在计算机中起着重要的作用，每一个寄存器相当于存储器中的一个存储单元，但它的存取速度比存储器（内存）要快得多。它用于存放计算过程中所需要的或所得到的信息，包括操作数地址，操作数据（原始数据、中间结果和结果数据）等。这些工作寄存器有时我们也习惯称它们为可编程寄存器，原因是这些工作寄存器绝大多数可用 8086/8088 指令在编程时对它们置数或读数。下面我们详细介绍这些工作寄存器。

## 1.2.2 8086/8088 的寄存器组

### 1. 数据寄存器（或称通用寄存器）

数据寄存器包括 AX、BX、CX 和 DX 四个通用寄存器，它们用来暂时存放运算过程中所用到的操作数、结果数据或其它信息。它们既可以以字（16 位）的形式使用，也可以以字节（8 位）的形式使用。

从图 1-3 可知，以字形式使用时，四个通用寄存器分别称为 AX、BX、CX、DX；以字节形式使用时，高 8 位通用寄存器分别称为 AH、BH、CH、DH。低 8 位通用寄存器分别称为 AL、BL、CL、DL。

这四个寄存器都是通用寄存器，但它们又可以用于专用的目的。

AX(Accumalator）作为累加器用，它是乘法运算中存放参加运算的一个操作数及存放运算结果数据——乘积或乘积的低 16 位部分。另外，所有的输入／输出指令都使用这一寄存器与外部设备传送信息。

BX(Base）可以用作通用寄存器。此外，在计算存储器地址时，它经常用作基地址寄存器，所以又称基址寄存器。

CX(Count)可以用作通用寄存器。此外，在循环（Loop）和串处理指令中用作隐含的计数器。

DX(Data)可以用作通用寄存器，在做双字长运算时，把 DX 和 AX 组合在一起存放一个双字长数。DX 存放高位字（高 16 位）。此外，对某些输入／输出操作，DX 用来存放 I/O 端口地址。

### 2. 指针及变址寄存器

它们包括 SP、BP、SI、DI 四个 16 位寄存器。它们可以像数据寄存器一样在运算过程中存放操作数，但它们只能以字(16 位）为单位使用。此外，它们更经常用于在段内寻址时提供偏移地址。

SP(Stack Pointer）堆栈指针寄存器：

用来指示堆栈的栈顶的偏移地址，与 SS 堆栈段寄存器一起形成栈顶存储单元的物理地址。

BP(Base Pointer) 基址指针寄存器：

用来指示堆栈中某个数据区的偏移地址——基地址，与 SS 堆栈段寄存器一起形成堆栈中某个存储单元的物理地址。

SI(Source Index) 源变址寄存器；

DI(Destination Index) 目的变址寄存器：

这两个寄存器与 DS 数据段寄存器一起用来确定数据段中某一存储单元的物理地址。这两个寄存器都有自动增量和自动减量功能，用于变址是很方便的。在串处理指令中，SI 和 DI 作为隐含的源变址寄存器和目的变址寄存器，此时 SI 和 DS 联用，DI 和 ES 附加段寄存器联用，分别达到在数据段中和在附加段中寻址的目的。

### 3.段寄存器

它们包括 CS、SS、DS、ES 四个 16 位的段寄存器。

CS(Code Segment) 代码段寄存器

SS(Stack Segment) 堆栈段寄存器

DS(Data Segment) 数据段寄存器

ES(Extra Segment) 附加段寄存器

8086/8088 采用存储空间的分段技术来解决寻址 1M 字节的存储空间。这些段寄存器的内容和有效的地址偏移量（称为偏移地址）一起即可确定内存的存储单元的物理地址。通常 CS 划定并控制程序区，DS 和 ES 划定和控制数据区，SS 划定和控制堆栈区，究竟如何确定内存单元的物理地址详见 1.3 节。

### 4.控制寄存器

控制寄存器分为两个 16 位的寄存器 IP 和 PSW。

IP(Instruction Pointer) 指令指针寄存器：

它用来存放代码段中的偏移地址。在程序运行过程中，它始终指向下一条指令的首地址。它与 CS 寄存器联合确定下一条指令的物理地址。当这一地址送到存储器后，控制器可以取得下一条要执行的指令，而控制器一旦取得这条指令，马上修改 IP 的内容，使它指向下一条指令的首地址。计算机就是用 IP 寄存器来控制指令序列的执行流程的，因此 IP 寄存器是计算机中很重要的一个控制寄存器。

PSW(Program Status Word) 程序状态字寄存器（或称标志寄存器）：

这是一个 16 位寄存器，由状态（或称条件码）标志(flag)和控制标志所构成，如下所示：

| 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 |

其中状态标志用来记录程序运行结果的状态信息。由于这些状态信息往往用作后续条件转移指令的转移控制条件，所以又可称为条件码。它包括以下 6 位：

OF(Overflow Flag) 溢出标志：

在运算过程中，如运算结果已超出了机器能表示的数值范围（指有符号数），称为溢出。此时 OF = 1 否则，OF = 0。

SF(Sign Flag)　符号标志：

记录运算结果的符号，结果为负时置 1 否则置 0。

ZF(Zero Flag)　零标志：

运算结果为零时，ZF = 1 否则，ZF = 0。

CF(Carry Flag)　进位标志：

记录运算时从最高有效位产生的进位值或借位值。最高有效位有进位或借位时，CF = 1；否则，CF = 0。

AF(Auxiliary Carry Flag)　辅助进位标志：

记录运算时第 3 位（半个字节）产生的进位值或借位值。第 3 位有进位或借位时，AF = 1；否则，AF = 0。这个标志用于十进制算术运算指令 DAA、DAS、AAA 和 AAS。

PF(Parity Flag)　奇偶标志：

用来为计算机串行传送时可能产生的代码出错情况提供检验条件。当结果操作数（一个字节数据或字数据的低 8 位）中“1”的个数为偶数时，PF = 1 否则，PF = 0。

控制标志位有三个：

DF(Direction Flag)　方向标志：

在串处理指令中控制处理信息的方向用。当 DF = 1 时，每次操作后使 SI 和 DI 减量，这样就使串处理从高地址向低地址方向进行；当 DF = 0 时，则使 SI 和 DI 增量，使串处理从低地址向高地址方向进行。

IF(Interrupt Flag)　中断标志：

当 IF = 1 时，允许中断；否则，关闭中断。

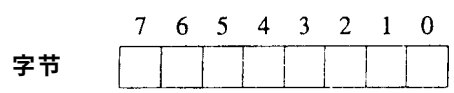
TF(Trap Flag)　跟踪标志：

用于程序调试时进行单步方式工作。当 TF = 1 时，每条指令执行完后产生一个内部中断，允许程序在每条指令执行后，可以进行检查；TF = 0 时，CPU 正常工作不产生内部中断。以上就是 PSW 中各种状态和控制标志的含义。其中控制标志是由系统程序或用户程序根据需要用指令来设置的　8086/8088 指令系统中提供了设置部分状态和控制标志的指令，给用户在程序中使用这些指令来设置某些标志。

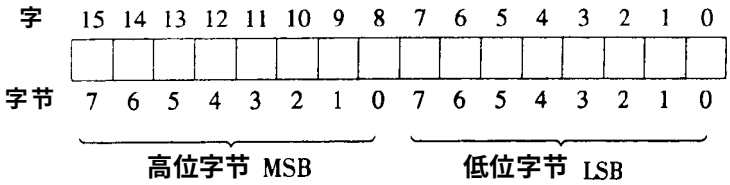
# 1.3　8086/8088 的存储器组织

## 1.3.1　存储单元的地址和内容

计算机存储信息的基本单位是一个二进制位，一位可存储一个二进制数 0 或 1。每 8 位组成一个字节。位编号如下所示：




8086/8088 的字长为 16 位，由 2 个字节组成，位编号如下所示：




在存储器里以字节为单位存储信息。为了正确地存放和取得信息，每一个字节单元给予一个存储器地址，地址从 0 开始编号，顺序地每隔一个单元加上 1。在计算机中，地址也是用二进制数来表示的。注意，它是一个无符号的整数，书写格式通常为十六进制数。

上节已提到，存放存储器地址的寄存器字长为 16 位。因此每个存储单元若用 16 位二进制数表示地址，那么 16 位的二进制数可以表示多少个存储单元呢？应该是 $2^{16} = 65536$ 个。在计算机里，为方便起见，总是以 $2^{10} = 1024$ 个存储单元作为存储器容量的一种基本单位，称为 1K(或 1K 字节)。这样 65536个存储单元的存储器存储容量就是 64K(或 64K 字节,或 64KB)。所以存储单元地址编号用十六进制数表示为 0000H ~ FFFFH。

一个存储单元中存放的信息（数据）称为该存储单元的内容。图 1 - 4 表示了存储器里存放信息的情况。

由图可知 4 号存储单元中存放的信息为 34H。也就是说 4 号存储单元中的内容为 34H,可表示为：

(0004H) = 34H

8086/8088 字长是 16 位，大部分数据都是以字为单位表示的。那么一个字（16 位）怎样存入存储器中呢？一个字存入存储器要占有相继的两个存储单元。存放时，字的低位字节存入低地址的存储单元，字的高位字节存入高地址的存储单元。也就是说，信息的存入次序和书写次序是相反的。一个字数据为 7B86H,那么存储时,先存入 86H,然后再存入 7BH。存储器中两个字节单元组成一个字单元，字单元的地址采用它的低位字节存储单元的地址表示。图 1 - 4 中 4 号字单元的内容是 1234H，可表示为：

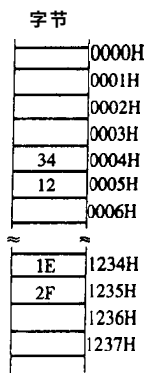



图 1 - 4 存储单元的地址和内容

(0004H) = 1234H

可以看出，同一个地址既可看作字节单元的地址，又可看作字单元的地址。那么这个地址究竟是字单元还是字节单元，如何区别呢？这在以后的章节中会详细说明，可在指令中定义操作数是字还是字节来规定。

一般来说，字单元的地址可以是偶数，也可以是奇数。但是，在 8086 中,访问存储器(取数或存数）都是以字单元进行的。也就是说，机器是以偶地址访问存储器的，对于要取一个奇地址的字单元中的数据，CPU 必须访问两次存储器，花费的时间要多一倍。

如上所述,如果用 X 表示某存储单元的地址，则 X 单元的内容可以表示为 (X),假如 X 单元中存放着 Y,而 Y 又是一个地址，则可用(Y) = ((X))来表示 Y 单元的内容，如图 1 - 4 中

(0004H) = 1234H

而 (1234H) = 2F1EH

则也可表示为：

((0004H)) = 2F1EH

存储器进行取数或存数，即读或写操作时，从某个单元中读出其内容后，该单元的内容不变，可以重复读出该内容；只有当写入新的内容，原来保存的内容就自动丢失。

## 1.3.2 存储器地址的分段

8086/8088 微处理器有地址总线 20 根，可访问存储器的最大容量为：

$$2^{20} = 1048576 = 1024K = 1M$$

所以用 16 进制数表示 1M 字节的地址范围为 00000H ~ FFFFFH。那么在 16 位字长的机器里，用什么办法来提供 20 位地址呢？在 PC 机里采用了存储器地址分段的方法。

8086/8088 的存储器组织是将存储器划分为段，每个段的大小可在 64K 范围内选取任意个字节，段内地址可以用 16 位表示(64K 范围内)。对段的起始地址有所限制，段不能起始于任意地址，而必须从任意一小段(Paragragh)的首地址开始。机器规定，从 0 地址开始，每 16 个字节为一小段，下面列出了存储器最低地址区的三个小段的地址区间，每一行写为一小段中的 16 个地址：

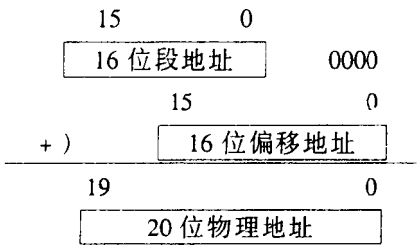
00000,00001,00002,……,0000E,0000F
00010,00011,00012,……,0001E,0001F
00020,00021,00022,……,0002E,0002F

其中，第一列就是每小段的首地址，00000H 为第一小段首地址，00010H 为第二小段首地址，00020H 为第三小段首地址。其特征是：在 16 进制数表示的地址中，最低位为 0(即 20 位地址的最低 4 位为 0000)。在 1M 字节的地址空间中，共有 64K 个小段首地址，可表示如下：00000H,00010H,……,FFFE0H,FFFF0H。

在 1M 字节的存储器中，每一个存储单元都有一个惟一的 20 位地址，称为存储单元的物理地址。CPU 访问存储器时，必须先确定所要访问的存储单元的物理地址才能取得（或存入）该单元中的内容。

存储单元的五位物理地址（16 进制数)是由 16 位段地址和 16 位偏移地址组成。段地址是指每一段的起始地址，由于它必须是小段的首地址，所以段的起始地址的低四位一定是 0000，因此就规定了段地址只取段起始地址的高 16 位值。偏移地址则是指在段内相对于段起始地址的偏移量，此偏移地址也是 16 位值。因此存储单元的物理地址的计算方法表示如下：

物理地址 = 段地址 × 16 + 偏移地址




换句话说，把段地址左移四位加上偏移地址值就形成物理地址。例如，某个数据存放在 DS = 8561H 和 DI = 3742H 的数据段的存储单元中，此存储单元的物理地址为：

85610H + 3742H = 88D52H

该存储单元的物理地址为 88D52H。

显然，每个存储单元只有惟一的一个物理地址，但它却可由不同的段地址和不同的偏移地址组成。例如，数据若存放在 ES = 7A62H, SI = E732H 的附加段的存储单元中，此存储单元的物理地址是 88D52H，因此还是那个存储单元。从此例可以看出，当某个单元段地址不变时，偏移地址也不会变。而该单元的段地址改变时，偏移地址也要变。

8086/8088 微处理器中，有四个专门存放段地址的寄存器。它们是代码段 CS、数据段 DS、堆栈段 SS 和附加段 ES 寄存器。每个段寄存器可以规定一个段的起始地址，每个段有各自的用途。代码段存放当前正在运行的程序，数据段存放当前运行程序所用的数据。如果程序中使用了串处理指令，则其源操作数存放在数据段中。堆栈段定义了堆栈的所在区域，堆栈是一种数据结构，它开辟了一个比较特殊的存储区，并以“后进先出”的原则访问这一区域，在下一章里我们还会专门讨论堆栈。附加段是附加的数据段，它是一个辅助的数据区，对于串处理指令，它是目的操作数存放的区域。当用户在编制程序时，应该按照上述规定把程序的各部分放在规定的区段之内，这在第 3 章中会详细说明。

除非专门指定，一般情况下，各段在存储器中的分配是由操作系统负责的。每个段可以独立地占用 64K 存储区，如图 1-5 所示。各段也可以允许重叠，下面的例子就可以说明这种情况。

例如，如果代码段中的程序占有 8K(2000H)存储区，数据段占有 2K(800H)存储区，堆栈段只占有 256 个字节的存储区。此时分段情况如图 1-6 所示。

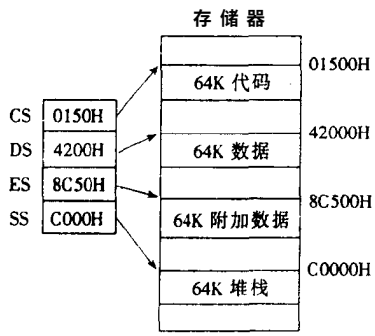



图 1-5 段分配方式之一

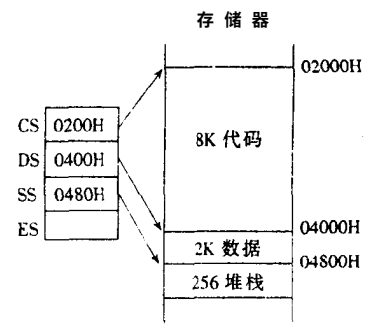



图 1-6 段分配方式之二

由图可知，代码段的区域本可为 02000H ~ 11FFFH(64K)，由于程序区只需要 8K 字节(02000H ~ 03FFFH)，所以程序区结束后的第一个小段的首地址（04000H）就作为数据段的起始地址。这样，代码段和数据段重叠在一起了。注意，每个存储单元的内容是不允许发生冲突的，也就是说，某个存储单元既存储了指令代码，又将存储操作的数据，这是绝对禁止的。这里所谓的重叠只是指每个区段的大小允许根据实际情况分配，而不一定非要占有 64K 的最大段空间，指定段区在内存位置的方法将在第 3 章伪指令语句中讨论。

如果程序中的四个段都在 64K 的范围内，而且程序运行时所需的信息都在本程序所定义的段区之内，那么，用户只要在程序的首部设定各段寄存器的值就可以了。如果程序的某一段(例如数据段)在程序运行过程中会超过 64K 空间，或者程序中可能访问除本身四个段以外的其它区段的信息，那么在程序中必须动态地修改段寄存器的内容，以保证所获得的信息的正确性，因此并不会因段区的划分而限制了程序空间的使用。这种存储器分段的方法虽然给程序

设计带来了一定的麻烦，但它可以扩大存储空间，而且对于程序的再定位也是很方便的。

## 1.4 外部设备

计算机运行时的程序和数据都要通过输入设备送入计算机，程序运行的结果要通过输出设备送给用户，所以输入输出设备是计算机必不可少的组成部分。大容量的外存储器（如硬磁盘和软磁盘）能存储大量信息，也是计算机不可缺少的一部分。对于外部设备的管理是汇编语言的重要使用场合之一。

从图 1-1 可见，外部设备与主机（包括 CPU 和存储器）的通信是通过外设接口电路进行的。在每个接口电路中都有一组寄存器。一般说来，这些寄存器有三种不同类型：

①数据寄存器：用来存放要在外设与主机之间传送的数据，这种寄存器实际上起缓冲器的作用。

②状态寄存器：用来保存外部设备或接口电路的状态信息，以便 CPU 在必要时测试外设或接口电路的状态，了解它们当前的工作状况。例如，每个设备都有忙闲位用来表明设备当前是否正在工作，是否有空接受 CPU 给予的新任务等。

③控制寄存器：CPU 给外设或接口电路的控制命令通过此寄存器暂存，适当时候向接口的有关电路或外设发出各种命令。例如，CPU 要启动磁盘工作，必须发出启动命令等。

当然外部设备通过接口与主机相连，每个接口中究竟需要如何设置这些寄存器以及这些寄存器的具体数量都是取决于不同类型的接口电路而定。这方面的内容由《微型计算机及其接口技术》课程中进行详细分析讨论，本课程不作介绍。

为使主机访问外设方便起见，外设中的每个寄存器给予一个端口（Port）地址（又称端口号），这样就组成了一个独立于内存空间的 I/O 地址空间。8086/8088 的 I/O 空间可达 64K 个端口地址，在 8086/8088 的输入／输出指令中间接寻址外设的地址总线使用了 16 根，所以端口地址的范围是 0000H ~ FFFFH。

主机与外设交换信息是通过输入／输出指令来完成的，在第 2 章中将介绍这部分指令。至于主机与外设的各种传送方式及其有关的汇编语言程序的设计，本书不作讨论，可参看《微型计算机及其接口技术》课程。

为了便于用户使用外设，PC 机提供了两种类型的例行程序（即子程序）供用户调用。一种是 BIOS(Basic Input/Output System)；另一种是 DOS(Disk Operating System) 功能调用。它们都是系统编制的子程序，通过中断方式转入所要的子程序去执行，执行完后返回原来的程序继续执行。

BIOS 和 DOS 功能调用虽然都是系统提供的例行程序，但是它们之间又有差别。BIOS 存放在机器的只读存储器 ROM 中，所以可以把它看成是机器硬件的一个组成部分，它的层次比 DOS 更低，更接近硬件，因此它的语句要完成每一个对设备的直接命令或信息传送。DOS 功能调用是操作系统 DOS 的一个组成部分，它在开机时由磁盘装入内存，在它的例行程序中可以一次或多次调用 BIOS 以完成比 BIOS 更高级的功能。有关这方面内容的详细讨论可阅读《操作系统》课程中有关章节，本书仅对子程序实例作些介绍，可参见后面有关章节。

# 5 汇编语言和汇编语言程序设计

## 1.5.1 什么是汇编语言

计算机的操作除了必须有 CPU 等硬件电路以外，还必须执行指令或程序。

指令就是使计算机执行各种各样操作的命令，这种命令由二进制代码来书写，如加、减、移位等操作命令，由于目前的数字式电子计算机是二进制的，它只能识别“0”或“1”以及它们的序列，所以这种二进制编码的指令是计算机惟一能识别和执行的指令。为了使计算机能完成人们预定的任务，必须执行一条条的指令，这种能实现一定任务的指令序列，称为程序。这种用二进制代码书写指令和程序的语言，称为机器语言。最早期的计算机所使用的编程语言就是机器语言。虽然这种语言书写的指令又繁琐又难以记忆，人们使用时很不方便，但这是计算机能直接识别的惟一的一种语言。为了克服难以记忆和书写繁琐，就发展了汇编语言。汇编语言用操作内容的英文词的缩写符号代替二进制编码，用符号代替地址或操作的数据，如用 AND 代替逻辑与等等。汇编语言书写的指令与机器语言书写的指令仍然是一一对应的。显然，汇编语言书写的指令易于为人们所理解和便于书写，但是汇编语言书写的指令和程序必须经过翻译程序——汇编程序翻译成二进制代码的指令和程序——目标程序，计算机对目标程序才能识别和执行。本课程的主要内容就是介绍汇编语言的指令、指令系统和汇编语言程序设计的一系列问题。

## 1.5.2 学习汇编语言和汇编语言程序设计的目的和意义

正如前述，汇编语言编写的程序可直接被计算机硬件识别和执行（当然，必须翻译为目标程序）。也就是说，汇编语言编写的程序是面向机器的。为了学好和掌握有关计算机的硬、软件知识，学习汇编语言和汇编语言程序设计有重要的现实意义。

1. 计算机的指令系统是计算机的重要性能特征

每台计算机都有自身的实现各种运算操作的命令，也即各种各样指令。一台计算机所具有的各种各样指令的集合，称为该计算机的指令系统。所以一台计算机指令系统的指令类型和数量多少，说明了该计算机运算处理能力的强弱。

以算术运算类指令为例，8 位机只有定点 8 位加、减运算指令，16 位机只有定点 16 位加、减、乘、除运算指令，而发展到 32 位机已有 32 位浮点运算指令。如果使用 8 位机或 16 位机实现 32 位浮点运算，必须要用二三十条加或减运算指令进行编程才能实现，而 32 位机只要用一条指令就可实现浮点运算。就存储字节数和运算速度而言，32 位机大大优于 8 位机和 16 位机。

所以为了了解一台计算机具有哪些类型的运算操作，尤其要选择符合我们应用系统所需

要的计算机，必须学习和熟悉计算机的指令系统中的各类指令。

2.学习与计算机硬件有密切关系的课程，必须具备汇编语言及汇编语言程序设计方面的知识

微型计 算机的工作就是取指令和执行指令。所以，微处理器必须经常与存储器或外部设备不断地交换信息（指令或数据）。也就是说，一台计算机的工作，硬件电路的运转必须配合执行指令和执行程序，两者缺一不可。

例如，学习到《微型计算机及其接口技术》课程，特别是有关接口电路内容的章节时，必须结合输入／输出指令的执行。在介绍接口电路的应用实例时，又有大量的用汇编语言编写的应用程序。其它如操作系统、微型机控制技术等课程中，也有大量涉及到汇编语言编写的指令和程序方面的知识。

3.汇编语言及汇编语言程序设计在微型机应用中占有重要地位

一般来说，凡是在微型机应用中涉及到与硬件电路有关的应用系统，如微型机控制系统中直接检测和控制设备部分，仪器、仪表中的计算机控制和自动数据处理部分，家用电器的计算机控制部分等，绝大部分都是用汇编语言来编写应用程序的。

当然，高级语言也可编制上述应用方面的程序（但是对外部设备的输入／输出程序，还得调用汇编语言编写的子程序），由于此类程序必须经过编译程序翻译成目标程序，由于编译的需要，高级语言编写的程序比汇编语言编写的程序要繁琐得多，占用的字节数要多得多。在单机控制，仪器、仪表及家用电器的控制方面，一般设备的体积都较小（电气控制部分），大部分藏在机器内部。因此，内存的容量不会很大，一般在几 KB 到几十 KB。所以，绝大部分都是用汇编语言来编写应用程序的。到目前为止，汇编语言编写的程序在微型机应用中仍占有很重要的地位。

## 小　　结

本章简要地介绍了计算机系统的基本概念和基本结构，阐述了系统的硬件和软件的组成，阐述了汇编语言程序设计在计算机应用中的重要作用和地位。

本章简要地阐述 8086/8088CPU 的编程结构， CPU 内部可编程寄存器组的分类、名称、符号、字长、功能和用途。

本章简要地介绍了 8086/8088 的存储器组织形式，存储器分段结构的意义和实现方法。阐述了物理地址、段地址和偏移地址的概念及相互联系，物理地址的形成方法。

本章简要地介绍了计算机外部设备及外部设备通过接口电路与 CPU 连接时,端口和端口地址的概念。

### 习　题　1

1.1　请将下列左边的项和右边的解释联系起来（把所选字母放在括号中）：

| | | |
|---|---|---|
| (1) CPU | ( ) | A. 存储程序、数据等信息的记忆装置。 |
| (2) IP | ( ) | B. 指出指令操作结果的标志，如 ZF、CF 等。 |
| (3) SP | ( ) | C. 是逻辑段的起始地址。 |
| (4) 状态标志 | ( ) | D. 分析、控制并执行指令的部件。 |
| (5) 控制标志 | ( ) | E. 保存当前栈顶地址的寄存器。 |
| (6) ALU | ( ) | F. 相对于段起始地址的偏移量。 |
| (7) 存储器 | ( ) | G. 指示下一条将要执行的指令的地址。 |
| (8) 物理地址 | ( ) | H. 控制操作的标志，如 DF、TF 等。 |
| (9) 偏移地址 | ( ) | I. 进行算术和逻辑运算的单元。 |
| (10) 段地址 | ( ) | J. CPU发出的访问存储器的地址信息。 |

1.2 下列操作可使用哪些寄存器：

(1) 存放各种运算操作的数据。

(2) 存放数据串操作时的计数值。

(3) 查看程序已执行到哪条指令的地址。

(4) 查看堆栈中当前正要进行入出栈的存储单元的地址。

(5) 查看运算结果是否等于零。

(6) 查看程序中的数据存放段区是从哪个地址开始的。

(7) 查看程序中的指令存放的段区是从哪个地址开始的。

1.3 段地址和偏移地址为 1000:117A 的存储单元的物理地址是什么？而 1109:00EA 或 1025:0F2A 的存储单元的物理地址又是什么？这说明了什么问题？

1.4 在存储器中存放的数据如下图所示。试读出 75422H 和 75424H 字节单元的内容是什么？读出 75422H 和 75424H 字单元的内容是什么？

存储器

| | |
|---|---|
| | ⋮ |
| 75420 | 13H |
| 1 | 78H |
| 2 | 9CH |
| 3 | 24H |
| 4 | 5DH |
| 5 | E6H |
| | ⋮ |

# 第 2 章　8086/8088 的寻址方式和指令系统

## 2.1　概　　述

众所周知，计算机必须具备 CPU、存储器和外部设备等硬件设备。但仅有这些硬件，还只是具有了计算和处理信息的能力。计算机能真正进行计算和处理还必须要有软件的配合，即各种程序(Program)。

人们要求计算机解决计算或处理信息的问题，首先必须把问题转换为计算机能识别和执行的一步步操作命令。我们把这种要求计算机执行的各种操作以命令形式写下来这就称为指令(Instruction)。通常一条指令对应着一种基本操作，例如加、减、传送和移位等等。一个计算机能执行什么样操作，能做多少种操作，是由该计算机的指令系统所决定的。因此，计算机所能执行的全部指令，就是计算机的指令系统(Instruction Set)。目前，微型计算机的指令系统可以包括几十种或百余种指令每种计算机都有自己固有的指令系统。8086/8088 的指令系统和 M68000 的指令系统具有不同的指令，不能相互兼容。也就是说，8086/8088 指令系统中的指令，只能由 8086/8088 微处理器所识别和执行，而不能被 M68000 微处理器所识别和执行。但 8086/8088 微处理器可以执行 8080/8085 指令系统中的指令，因为 Intel 公司设计的 Intel 8086/8088 16 位微处理器的指令系统对它的 8 位产品 Intel 8080/8085 的指令系统是向上兼容的缘故。

在使用计算机时，必须把要解决的问题编成一条条指令，当然这些指令必须是我们使用的计算机所能识别和执行的指令，也即每一条指令必须是一台特定计算机的指令系统中所具有的指令，而不是随心所欲，信手写来的指令。我们把这一条条指令的集合或指令序列叫做程序。用户为解决自己的问题用各种语言所编写的程序，称为源程序。

计算机中的指令由操作码字段和操作数字段两部分组成。操作码字段指出计算机所要执行的操作，而操作数字段则指出在指令操作过程中所需的操作数据。例如，加法指令一方面需要指定做加法操作的部分，即操作码字段。另一方面需给出被加数和加数部分，即操作数字段。操作数字段可以是操作数本身，也可以是操作数地址或地址的一部分，还可以是指向操作数地址的指针或其它有关操作数的信息。指令的格式一般是：

操作码　　操作数，……，操作数

操作数字段可以有一个或两个，通常称为一地址、二地址指令。例如，单操作数指令，它只需指定一个操作数参加操作，如移位指令、增 1、减 1 指令等，这就是一地址指令。大多数运算指令是双操作数指令，如算术和逻辑运算指令。两个操作数为源操作数和目的操作数。尽管在指令执行前这两个操作数都是原始操作数，但指令执行后将把运算结果存放到目的操作数

的地址单元中去，当然目的操作数的原始数据将会丢失。如果此原始操作数在以后的运算中还会用到，那么必须在进行运算前，给它准备一个副本（即预先存储在内存或寄存器中）。8086/8088 的运算指令就采用这种二地址指令。

指令中操作数字段实质上指出参加操作运算的操作数存放于何处。一般说来，操作数可以存放在指令代码中，称为立即数。操作数也可存放在 CPU 的内部寄存器中，称为寄存器操作数。操作数绝大部分是存放在内部存储器中，称为存储器操作数。对于一部分输入输出指令来说，操作数可以存放在接口电路的寄存器中。指定立即数和寄存器操作数的表示方法比较简单。而对存储器操作数来说，一个存储单元的地址就需要 20 位，怎样设法使它在指令的操作数字段的表示中减少位数呢？另外，从程序运行时的数据结构来看，操作数常常不是单个的数，往往是成组的以表格或数组形式存放在存储器的某一区域中，在这种情况下，指令用什么方式来指定操作数的地址更好呢？从程序设计的通用性来看，操作数或操作数存放的地址在指令中的指定应具有易于改变的灵活性，需要有多种方式来指定操作数或操作数地址。指令中用于说明操作数所在地址的方法，称为寻址方式（Addressing mode）。

我们知道，计算机只能识别二进制代码，机器指令是由二进制代码组成的，这种指令称为机器码。机器码是由一连串的 0 和 1 组成的，没有明显的特征，人们不好记忆，不易理解，易出错。所以，编制程序成为一种十分困难和繁琐的工作。因而，人们就用一些助记符（Mnemonic），通常是指令功能的英文词的缩写来代替操作码，如 8086/8088 中，数的传送指令用助记符 MOV（MOVE 的缩写）表示。这样，每条指令有明显的特征，易于理解和记忆，也不易出错，此即汇编语言指令。汇编语言书写的指令操作码用助记符代替，操作数也可用符号（Symbol）或符号地址（称为标号）来表示，它与机器指令是一一对应的。本书均用汇编语言格式来书写指令。

## 2.2 8086/8088 的寻址方式

### 2.2.1 有效地址 EA 和段超越

当操作数是存放在存储器中时，存储器的存储单元的物理地址有两部分组成。一部分是偏移地址；一部分是段地址。在 8086/8088 的各种寻址方式中，寻找存储单元所需的偏移地址可由各种成分组成，称为有效地址，用 EA 表示。不同的寻址方式，组成有效地址 EA 的各部分内容也不一样，详见以下各节讨论说明。

存储器操作数寻址时，存储单元的物理地址的另一部分是段地址，对段地址是如何规定的呢？8086/8088 指令系统中对段地址有个基本规定，即所谓 Default 默认）状态。在正常情况下，由寻址方式中有效地址规定的基地址寄存器来确定段寄存器，即只要寻址方式中出现 BP 寄存器作为基地址，段寄存器一定采用堆栈段 SS 段寄存器，其余的情况都采用数据段 DS 段寄存器。串处理指令有另外规定，详见串处理指令一节说明。

指令中的操作数也可以不在基本规定的段区内，必须在指令中指定段寄存器，这就是段超

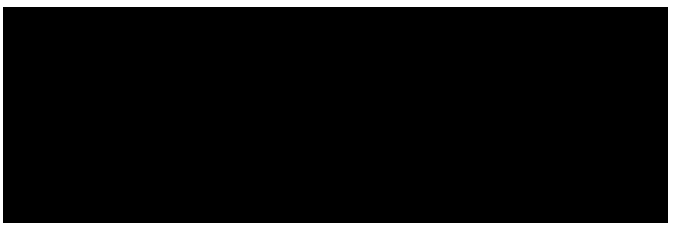
越。例如：

此存储单元的物理地址为 ，数据是存放在数据段中。

而 ：

现在存储单元的物理地址为 ，此指令传送的源数据却在附加段中。段地址的基本规定（或约定）和允许超越的情况如表 所示。

存储器存取时的约定段和可修改段



## 与数据有关的寻址方式

立即寻址方式（

操作数直接存放在指令中，紧跟在操作码之后，它作为指令的操作数字段存放在指令代码中，这种操作数称为立即数。立即数可以是 位的或 位的。如果是 位立即数，则低位字节数存放在低地址单元中，高位字节数存放在高地址单元中。机器码存放形式如下所示：

操作码
立即数

立即寻址方式的操作数用来表示常数，它经常用于给寄存器赋初值，并且只能用于源操作数字段，不能用于目的操作数字段。

【例
指令执行后， 位数据 存入 寄存器。操作的示意图如图 所示。

【例
指令执行后， 位数据存入 寄存器。操作的示意图如图 所示。

操作码
操作数
（立即数）

图 操作示意图 图 操作示意图

图 指令存放在代码段中， 表示该指令的操作码部分，接下去存放 ,再存放这是立即数，它是指令机器码的一部分。

【例 2-3】 MOV BL,COUNT

在汇编语言的指令中，可以用符号 COUNT 代替常数，但是 COUNT 必须用伪指令 EQU 来赋值，详见第三章伪指令一节。

2.寄存器寻址方式( Register Addressing)

操作数在寄存器中，指令指定寄存器号，对于16位操作数寄存器可以是 AX、BX、CX、DX、SI、DI、SP 和 BP 等；对于 8 位操作数，寄存器可以是 AH、AL、BH、BL、CH、CL、DH 和 DL 等。这种寻址方式由于操作数在寄存器中，不需要访问存储器来取得操作数，因而可以取得较高的运算速度。这种寻址方式寻找操作数的示意图如图 2-3 所示。

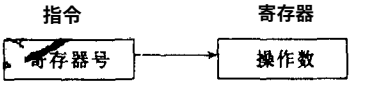



图 2-3 寄存器寻址方式示意图

【例 2-4】 MOV AX,BX

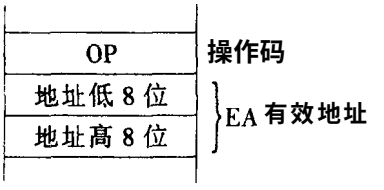
指令执行前，AX = 3064H，BX = 1234H。

指令执行后，AX = 1234H，BX = 1234H。

除上述两种寻址方式外，以下各种寻址方式的操作数都存放在代码段以外的存储器区段中，通过采用不同方法求得操作数地址，从而可取得或存入操作数。

3.直接寻址方式（Direct addressing)

在直接寻址方式中，有效地址 EA 就在指令的代码段中，它存放在代码段中指令操作码后面的操作数字段，机器码可表示如下：




此寻址方式与立即数寻址方式相比较，在指令的代码中，直接寻址的操作数字段是偏移地址，立即数寻址的操作数字段是立即数。由于操作数在正常情况下是存放在数据段中，所以必须先求出操作数的物理地址，然后再按照此地址访问存储器才能取得数据或存入数据。这种寻址方式寻找存储器的操作示意图如图 2-4 所示。通常操作数的物理地址为：

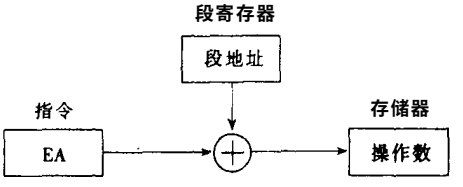



图 2-4 直接寻址方式示意图

$$物理地址 = DS \times 16 + EA$$

8086/8088 允许数据存放在数据段以外的其它段中，此时在指令中应指定段超越的段寄存器名，或称为指令应加上段超越前缀，在计算物理地址时，应使用指定的段寄存器，段超越可指定的段寄存器，可参看表 2-1。

【例 2-5】 MOV AX,[2000H]

如 DS = 3000H，从指令可知，EA = 2000H，存放源操作数的存储单元的 物理地址 = 30000 + 2000 = 32000H。