第一章　程序开发的系统观念

众所周知，现代计算机系统是由硬件与软件组成的一个统一整体，硬件是软件运行的物质基础，软件使硬件得以发挥作用，是系统的灵魂。软件与硬件的这种相互依存与补充的关系不仅仅是逻辑上的，也体现在计算机系统工作的任何时候，包括运行自己开发的程序的时候。因此，在开发程序时，不仅应考虑所写程序能够静态地实现预定的功能，还应考虑是否充分利用了系统的软硬件资源，发挥系统所具有的潜力。

1.1　程序运行的系统特征

假如我们用高级语言写好了一个程序，又经过编译和连接，形成了一个可以运行的目标程序，那么，这个程序是怎么运行的呢？

每个可运行程序都是以文件形式存放在磁盘上的，在它被加载到内存之前，只不过是磁盘上有名字的某组信息而已，与其它各式各样众多的文件共存于磁盘上，依靠操作系统统一管起来。操作系统按层次目录结构登记所有文件的文件名，登记在磁盘上由操作系统分配的各自占用的盘簇(块)，标记文件类型，用户存取权限，建立与修改的时间信息等。操作系统的文件管理系统使得我们的程序不被别人写磁盘时冲掉，又能简单方便地对文件按名存取，而用不着去关注磁盘文件的存放位置与磁盘物理结构。

操作系统为用户提供了加载并运行程序的接口，最简单的就是在命令状态下，允许用户发一条命令启动程序运行。比如在 DOS 环境下，如果可运行程序驻留于 C 盘当前目录之下，文件名为 myprog.exe，则在命令状态下，可依下述方法启动它

C>myprog

此后系统又会有一系列软、硬件动作，要检查文件大小，申请、分配内存，读盘，将文件调入内存，再启动程序的第一条指令，开始程序的顺序执行。

源程序中的每一条语句都将由目标程序中的一系列机器指令实现。我们先来看看一条指令的执行过程。

每一条典型的指令都可分解为一串操作序列，这些操作序列按时序又可分为指令周期。一条指令的执行过程通常可分为：①取指令周期，②取操作数周期，③执行周期。具体地说，取指令周期的执行过程是，指令地址送往指令地址寄存器，再送往存储器的地址寄存器，从存储体中取出指令，将指令送往 CPU 的指令寄存器。在这个过程中，同时完成 PC 值加 1，以便为执行下一条指令做好准备。取操作数周期的执行过程与指令类型有关，比如，运算类指令一般要经历两遍，第一遍完成从存储器取源操作数，第二遍从存储器取目的操作数，将它们与指令码一起送往指令寄存器。在指令的执行周期中，首先要对指令进行译码，然后发一系列微操作信号，控制全机有关硬件部分协调执行。运算指令的运算要在算术逻辑部件 ALU 中完成。

每一条指令执行过程的不同阶段，均可能要多次地与 CPU 和内存打交道，依指令的类型不同，也可能还要驱动键盘、显示器、鼠标、硬盘、软盘，打印等外没动作，这时从外界就容易看到计算机确实在忙碌地工作。

从使用的角度看，尽管一条指令的执行包含了复杂的硬件动作，仍然可把它当作“原子”看

待。然而，在一条指令执行完之后，系统将可能出现预料不到的情况。通常情况下，一条指令执行之后，可能的后继动作是：①顺序执行程序中下一条指令。这儿的'下一条指令'，包括程序中循环，转移结构含义下的下一条指令。②处理外部事件。由于系统中有多种外部设备，这些设备都有独立工作的能力，但必须靠CPU程序去启动和管理它们。外设以中断方式与CPU通讯，中断就是一类外部事件，为使外设能得到利用，CPU必须及时处理各种外部事件。③处理主机内部事件。CPU如果执行了一条非法指令，比如访问了非法地址，或者做除法时溢出，电源掉电等，也可以看成是"事件"，它是由主机内部产生的。有趣的是，某些非法指令，是由操作系统提供，故意让用户去使用的。这种非法指令叫广义指令或系统功能调用。这种指令一旦执行，系统将产生"自陷"(trap)，CPU转去执行一段操作系统内核的程序，达到使用某种系统资源的目的。

外部事件和内部事件的发生，将可能使得程序的运行出现很复杂的情况。CPU启动外设从事某项操作，比如要打印机打印一个字符后，继续做别的事，外设在与CPU工作的同时，完成这一单调的工作，做完了便向CPU发一个中断信号，以便得到CPU的又一次关注。CPU在执行完一条指令后，只要没有屏蔽中断，就会暂停当前程序下一条指令的执行，转去执行一段中断处理程序，进行"中断处理"，完后再恢复当前程序运行。

外部设备给CPU的中断信号，对于主机而言，就是一种外部事件。外部事件的发生是随机的，因为程序运行中无法预知某一事件的发生，比如是否有某个按键被按下了，而这个按键动作可能与当前程序毫不相干。又由于系统中有多台外设，故这种外部事件可能有多个，也可能同时发生。操作系统为了尽可能发挥各种外设的效率，会对这类事件进行排队管理。

在主机上运行的程序将可能随时被打断而转向中断处理，此后在适当时候再恢复运行，使得程序的执行处于某种走走停停的状态，在多任务环境下更是如此。

以上分析说明，程序运行不是孤立的。任何程序，也包括我们自己开发的程序，哪怕是一个小程序，都紧密地依赖硬件及支持程序运行的整个系统的软件环境。换句话说，程序必须在计算机软硬件运行环境的支持下运行。快速的CPU，快速且大容量的内存，良好的外设，系统提供丰富的事件处理功能，高效与可靠的事件处理策略等都是程序运行环境的组成要素。

常常有这种现象，某个程序只能在640k以上内存的机器上运行，否则，若拿到一台小容量的机器上，便会被告知"内存不够"而拒绝执行；有的程序在这个DOS版本下运行正常，换一个操作系统便不能正确运行，得到的报告是"DOS版本不对"。这是很容易体验到的事实，也是关于程序运行依赖于系统运行环境的很好的说明。

对于一个系统程序员，应有明确的观点，知道程序运行紧密地依赖于系统运行环境，深入地去熟悉它，在编写程序时，自觉地使程序适应当前机器的软硬件运行环境，才能写出高效而又有较好兼容性的程序。

1.2 程序开发的系统支撑环境

了解了程序运行依赖于系统环境后，我们来讨论，怎样把一个程序编写出来，或者开发出来？

一、程序开发步骤

按照软件工程观点，可以用"七阶段模型"描写开发过程，即

(1)开发项目的目标分析

(2)程序功能说明

(3)系统环境选择,包括硬件支持环境,所用的编程语言,软件开发工具选择,关于人力资金的说明等。

(4)系统设计

(5)系统构成与编写

(6)测试与调试

(7)运行与维护

由此看出,开发一个程序不单是指在一台已有机器的特定条件下的程序编制,比如在IBMPC 386机上用BASIC写一个应用程序,也包括程序运行的软、硬件支撑环境的选择,甚至包括机器选型,用什么语言,单用一种语言还是要用两种以上语言,两种语言之间是分别生成可执行代码,用在程序中执行另一个程序的方法连接,还是以混合语言编程,内嵌式调用等等。

二、软件开发环境

软件开发环境与工具的使用贯穿到程序开发的各个阶段,在设计阶段,要用结构化程序设计方法把程序分层分块,确定块间联系,确定数据结构,画出块内的程序流程图。

编写阶段的任务是为每个模块编写程序,也就是将模块说明书转换成用某种程序设计语言写的源程序。而且应当获得经编译通过的达到预期运行功能目标的程序。

1. 编辑程序

最常用的编程工具之一是编辑程序,它提拱支持,让我们把程序和数据结构写到磁盘文件中。常用的有行编辑和全屏幕编辑两大类,而以全屏幕编辑更直观和易于使用,最为流行的行编辑程序有DOS环境下的edlin, unix下的ex,ed。屏幕编辑程序有DOS下的word star, DOS 5.0之后的edit, unix的Vi,中文DOS环境下的wps,在各种窗口环境下,如微机windows系统,工作站的open window, sunview,都有很好的全屏幕与多窗口编辑环境,在多种计算机语言集成开发环境中,如C ,PAscal, BASIC语言也都有很强的全屏幕与多窗口编辑环境。

任何人写程序都不可能没有错,正像再好的作家也难以不加修改地写出一篇好文章一样,程序也必须边写边修改。因此好的编辑程序对于准确快速地写出程序是很重要的。

2. 编译与连接程序

有了源程序,下一步就是编译,将高级语言程序翻译成机器能够运行的形式,用编译程序对程序作调试性运行以便发现错误,在七八十年代,人们可能用优化与不优化的两种不同的编译程序。首先用快速但不优化的编辑程序,这种编译程序能很快生成目标程序,但是生成的目标程序可能在存储消耗和执行速度上效率都是相当低的。一旦程序通过调试准备投入运行时,采用一个优化编译程序来生成高效的机器代码,优化编译程序运行速度较慢,但它所生成的目标代码质量很高。现代计算机系统由于CPU快,内存容量大,故在集成环境下的编译程序一般都是优化的。

在八十年代以前,人们普遍认为一位优秀的汇编程序员能够生成比优化编译程序好得多的代码。今天的优化编译程序已经达到了这样高的水平,它们所生成的代码在质量上已经达到甚至超过了高度熟练的汇编语言程序员。像操作系统那样对效率要求很高的程序,已经不再非用汇编语言写不可了。大多数典型的现代操作系统是用C语言编写,并经高质量优化编译

程序翻译成高效机器代码的。

形成能够运行的代码过程中，实际上还要用到连接程序，不过有的系统把 link 功能与编译集成在一起，用户不必单独发连接命令。连接程序的作用是把程序中调用的函数及标准的程序代码嵌入到调用程序中。

无论在编译，连接还是在运行过程中，都可能发现程序有错误，因而往往要回到，甚至多次回到对源程序的编辑。

3. 调试程序

调试程序是最重要的程序开发工具之一，比如 DOS 环境下的 debug，UNIX 环境下的 adb 等，对于任何一个从事系统程序开发的人员来讲，掌握它有着特殊的意义。

调试程序可以对被调试的程序进行试运行。运行方式可以是单步的，即从当前给定的指令地址起，只执行一条指令；也可以是连续执行某一段程序。当然，如果程序没有错，可以一直执行下去的话，也可以从任意指定的指令开始，比如从程序开头起，运行整个程序，直到最后一条指令。无论执行一条指令，还是执行一个程序段，都允许设置程序运行的初始环境。可以设置“断点”，以限定要调试运行的程序段。

每启动试运行一次，debug 都会报告被调试程序运行至预定目的地时的运行状态，包括 CPU 的各个通用寄存器的值。例如在 DOS 环境下的 AX，BX，CX，DX，CS，ES，SS，BP，SP 以及标志存器的值，报告程序运行至此的中间结果，这对于发现并改正程序编制中的错误大有好处。

调试程序通常还有很多其它功能。诸如，辅助输入功能，输入可运行程序，且对输入语句的语法加以检查，拒绝接受错误语句，遇此情况便提示用户重输；模式匹配，字串搜索功能；文件操作功能；内存地址定位与内存读写功能；多种字符编码的转换与显示功能等等。由此可以看出，掌握调试程序的运用也是系统程序员的基本功。

4. 操作系统

操作系统是系统软件资源的管理者，它的资源管理功能使得系统资源利用率提高和方便用户对资源的使用，处理各种事件，组织系统中各种程序运行流程，使得系统得以有条不紊地工作。上一节讨论了，程序运行过程中与操作系统的工作是紧密相关的。从程序开发的角度讲，操作系统也是最基础的软件。

操作系统向用户程序提供了使用系统资源的接口——系统调用。每一系系统调用都是操作系统内核的一段程序，完成对某种资源的使用，比如键盘输入，打印机输出，磁盘操作，文件读写等。这些操作如果不经操作系统支持，将或者不可能进行，或者很困难。有了系统调用后，用户程序只要准备好必要的入口参数，就可以象子程序调用那样引用系统调用。在形式上，系统调用出现在用户程序中就像普通机器指令一样，它只占用户程序一条指令的空间。系统调用是操作系统级程序开发环境很重要的组成部分。

作为一个程序员，为了开发高效而简洁的程序，必须熟悉系统调用的功能和用法。并有足够的应用实践。

操作系统提供的程序开发环境还有若干其它方面，诸如提供并安装设备驱动程序，键盘按键的重新定义，显示方式的设置，文件的命名及层次目录结构，搜索命令与数据文件的路径，批文件，命令提示符的设置，字符编码转换，内存缓冲区个数的设置，打开文件个数的约定等等。作为从事面向系统的程序开发来说，掌握并恰当地运用它们，无疑同样是十分重要的。

1.3 系统开发的目标

进行系统的程序开发就是要写出符合要求的高质量的程序，具体目标是：①可靠性；②适应性；③效率；④理解性。

一、可靠性

可靠性一般包括正确性与坚定性两方面。

正确性的含义是指软件系统本身没有错误，在预期的环境下能正确完成预期的功能；坚定性是指万一硬件发生故障或输入数据不合理等意外环境条件下，系统仍能适当地工作。

对于一个小型程序，可以要求它是完全正确的，有时也可以证明其正确性；而一个大型程序的正确性是难以保证的，因为它的环境条件很多。例如可能有多种多样的硬件故障。输入数据可能有这样那样的奇异性，而在调试阶段不可能把各种各样条件逐一组合令程序试运行。例如 IBMOS/360 系统每个新版本的推出都改掉原有版本的上千个错误。虽然不能期望一个大型程序在任何情况下都是正确的，但是一个坚定的程序在遇到意外情况时，能按某种预期的方式作出适当处理，不至于丢失重要信息，避免造成灾难性后果，也能较快地从借误中恢复。

从概念上讲，正确性与可靠性并不等价，如果一个完全正确的程序，它不检查输入数据的合理性，若遇到奇异的输入，仍然可能造成严重后果，因而这个程序是不可靠的。而一个可靠的程序却不一定是正确的，我们的目标是追求一个基本正确的可靠程序。

二、适应性或者可维护性

适应性包括三方面含义，可扩充，可移植和可修改。开发的程序在运行阶段仍然可能错，需要修改；用户也可能有新的要求，需要对程序加以扩充；也可能需要将程序从一种机型移植到另一种机型。

维护工作也是相当困难的，有人统计，如果程序员一次修改 5—10 个语句，则修改成功的可能性是 50%，而一次修改 40～50 个语句，则成功率下降至 20%。为了使程序容易维护，在开发早期就必须采用一定的技术仔细分析，精心设计，并建立完整的文档。

三、效率

效率是指系统是否能有效地使用计算机资源。如 CPU 资源，设计中断处理程序，前后台或多任务可提高 CPU 和内存的利用率；利用高位内存，扩展内存可以使内存资源得到更充分的利用；用假脱机技术可提高打印机的共享率；用磁盘压缩技术，可提高磁盘容量等等。过去硬件价格昂贵时效率问题一向是非常强调的，随着硬件价格大幅度下降，近年来，已不像过去那样追求效率了。与追求易维护性、可靠性等往往是相互抵能的，片面地强调节省时间和空间，设计出来的系统就可能结构软复杂，难以理解和修改，追求可靠性一般也需要以一定时间和空间作为代价，在硬件价格下降，人工费用上升的情况下，目前人们宁可牺牲一点效率而获得较好的易维护性和可靠性。

四、清晰性与易理解性

新开发的程序不仅仅提供机器执行，还要经常供人阅读，首先是开发者本人在软件生命期的各个阶段都要阅读，所以质量好的程序必须是容易理解的，容易理解也是容易维护的前提。

易理解性可以有两重含义，一是指系统内部采用结构化程序设计方法，模块划分合理，层次清晰，对于软件人员易于阅读和理解，二是指系统人一机界面简明清晰，对于用户来说易于接受和使用。

综上所述，一个程序系统应保证高可靠性，好的可维护与适应性，高效率和清晰与易理解性，对于面向系统的程序设计而言，往往要为新的高层用户提供新的开发环境，质量问题将更显得重要，从开发过程早期开始，就应该注意采用一定技术来保证。

1.4 程序开发人员的素养

从事面向系统的程序开发，必须具备下述条件：

(1)熟悉系统的硬件结构，工作原理，会对有关硬件进行编程控制，比如在 PC 系列微机环境下，了解 CPU 类型及其指令序列，内存容量，是否有 Cache，是否有扩展内存，中断控制器 8259，可编程外围接口 8255 的结构统一编址的内存地址与编程控制；串/并行通讯硬件及通讯编程；打印机控制序列的使用。这是完成进一步编程的基础。比如，欲编写一个打印机驱动程序，就必须熟悉打印机工作原理及有关的控制序列。

(2)熟悉以操作系统为核心的软件开发环境，熟悉系统调用的使用。因为直接用端口地址进行硬件编程，虽然效率高，但是很麻烦，而且有时候做不到，比如系统未提供那种硬件的使用接口。又比如文件操作，必须通过系统调用，单纯硬件操作是不行的。

在 PC 系列微机环境下，除了系统调用，还有更低层的 BOIS 调用。从软件分层角度而言，它介平系统调用与硬件编程之间。如果遵从经典的定义，操作系统是最靠近硬件的最底层的软件。那么 BIOS 调用也可以划至操作系统；如果把 BIOS 调用看作固化了的软件，则也可以把它当作硬件看待。

除了操作系统环境，还有上一节讨论的编辑程序，连接编译程序，调试程序等等都应该掌握它们的使用。

在软件开发环境中，熟练地掌握汇编语言和 C 语言，PASCAl，BASIC 等高级语言的编程技术是最起码的基本功。汇编语言和 C 比较适合于面向系统的程序编制，著名的 UNIX 操作系统就是用约 90%的 C 语言代码和 10%的汇编语言代码编写的。其实每一种语言都在发展，不断吸收其它语言的长处，尽可能地适应硬件与其它软件的发展，比如 BASIC 语言就经历过 BASIC—BASICA—true BASIC 与 Quick—BASIC 等多个发展阶段。它也有系统开发所特有的一些功能，如内存直接读写，中断程序设计，调用汇编语言程序，端口编程，机器之间通讯，绘图与发声功能等。因此，对于语言工具的掌握，应一专多能，必要时能做到“杀鸡用鸡刀，杀牛用牛刀”，用得纯熟得当。

(3)熟悉系统开发的常用技术

面向系统的程序开发技术有许多方面，主要包括：

①直接对硬件编程，当特别追求程序效率时，比如驱动某台设备与主机交换(输入/输出)数据，就可能要直接通过端口查询设备状态，控制与驱动设备进行传输。

②缓冲技术。当需要在主机与外设之间交换信息时，由于外设与主机速度相差甚远，为使主机摆脱慢速外设的拖累，必须开辟内存缓冲区。

③中断程序设计。中断是外设与 CPU 之间的一种通讯手段，也是 CPU 之间的一种通讯手段，也是 CPU 处理各种各样内部与外部“事件”，合理组织计算机工作流程的基础，面向系统的程序有相当一些是以中断方式启动执行的。进行面向系统的程序设计通常要编写中断处

理程序，保留与修改中断向量表。

④内存驻留技术。让系统程序常驻内存，可以省去启动程序执行的许多时间。通常执行一个程序时，需先搜索程序所在的磁盘文件，为文件分配内存，启动磁盘传输，将文件读入内存，然后才能启动执行。如果程序已驻内存，则前面这许多时间都可节省下来。流行的CCDOS操作系统的字库及汉字输入与显示程序就是常驻内存的。

⑤扩展内存的使用。随着应用程序功能增强，尺寸增加，640K常规内存逐步显得不够用，因此，现在有些系统程序把自身驻留在扩展内存，以便不占或少占常规内存。

⑥信息压缩技术。为了节省存储空间，提高信息传递速度而对信息进行压缩，也是被经常作用的技术之一。比如已广泛流行的多媒体技术，面对大量的图像及语音信息，必须进行压缩，否则空间开销将不堪设想。只有有效的信息压缩技术，多媒体技术才能得到实际的应用与推广。

本书将以实例介绍上述技术的实现，这也正是本书重点讨论的内容之一。不过对于信息压缩，扩展内存的使用的用例基本上是原理性的。

(4)有软件工程知识，能够用软件工程的观点和规范指导程序开发。会用结构化程序设计的思想，比如软件生命期的概念，由上到下，逐步求精原则，贯穿于程序开发的全过程。本书并不去讨论软件工程规范，而着眼于具体的程序开发技术。但程序开发者必须了解程序设计在整个软件开发过程中所处的地位，了解编程之间和编程之后要做些什么工作。

小　结

所谓面向系统的程序开发，我们把它理解为比较接近软件底层，能比较直接地充分利用系统软、硬件资源，较好地运用以操作系统为核心的软件开发环境的程序开发。这种程序开发的目标是追求高可靠性，高效率以及易移植易理解。

程序的运行有很强的系统特征，它牵涉到系统中的操作系统及有关程序以及硬件的协调动作。程序的开发也依赖于系统的一整套开发环境。因此，从事面向系统的程序开发，必须熟悉系统中软、硬件的结构与工作原理，熟悉程序开发环境与工具的使用，了解软件工程的思想与方法，掌握并认真去实践诸如直接硬件编程，中断程序设计，缓冲技术，内存常驻技术，扩展内存使用，串并行通信，信息压缩技术，以及各种设备的使用等技术。

本书以下各章我们将围绕着系统中各种软、硬件资源的使用，以及上述各项程序开发技术展开讨论，并给出典型程序以供实践。

第二章　基于机器硬件与操作系统核心接口的程序开发

计算机硬件与操作系统核心接口都是可编程利用的资源。这些资源包括：

(1)机器硬件

(2)系统数据

(3)BIOS 调用

(4)软中断调用

(5)操作系统功能调用

在多用户系统中，用户并不能直接访问系统数据和机器硬件。因为不经操作系统协调，直接改写系统数据或对硬件进行操作，将可能给系统工作造成灾难。故用户程序必须经系统调用。任何越过操作系统直接写内存，读、写磁盘，驱动打印机，修改系统数据的企图都会遭到操作系统的拒绝。

为提供用户程序使用系统资源的条件，操作系统提供了一整套系统调用。本章我们将以主要精力集中在系统调用的使用上。

操作系统提供的系统功能调用，在 UNIX 中被称为系统调用，是操作系统内核的对外接口。在本书以下的叙述中我们把功能调用和系统调用作为同义词看待。用户程序，包括系统提供的命令，归根到底都调用了系统提供的服务来实现自己的功能。系统调用是操作系统与用户程序的接口。其它软件之所以说要操作系统支持，本质上是因为它们调用了操作系统的系统调用。因为系统资源是由操作系统管理的，故用户程序使用资源就必须向操作系统申请，其界面就是系统调用。

对于大量使用的微机而言，操作系统在提供一批系统功能调用的同时，也允许用户程序直接访问硬件和系统数据，在 DOS 环境下，还提供了比功能调用更低层，更接近硬件的服务，称为 BIOS 调用与软中断。它们提供了访问 IBM PC 及兼容机硬件，包括从键盘读取字符，把信息显示于屏幕上，读写磁盘，将字符送打印机打印和许多其它服务。

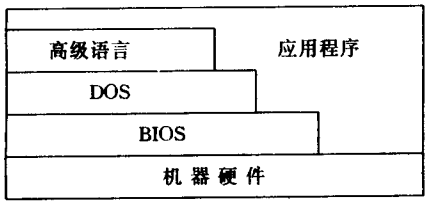

图 2.1　控制 PC 硬件的四种方式

一般来说，程序可以通过四种方式控制 PC 硬件，如图 2.1 所示。即直接访问硬件，使用 BIOS 功能，使用 DOS 功能调用和使用高级语言提供的功能，四种方式在编程的复杂性和程序的可移植性方面各有利弊。

从事面向系统的程序开发，应该掌握利用系统资源的各种方法，在实际运用中加以取舍。通常情况下，我们提倡用高级语言编程，比如用 C 语言，通过 DOS 功能调用使用系统硬件，必要时，也要用 BIOS 调用和直接使用硬件。这就给程序员提供了广阔的活动舞台，当然也就提出了更高的要求。下面我们就从 PC 环境开始讨论。

2.1 与机器硬件打交道

机器硬件包括 CPU、内存、键盘、显示器、磁盘、打印机等外设，直接使用它们一般要用机器硬指令。

一、控制 CPU

CPU 是运行程序的，程序中的语句序列便是对 CPU 的控制。有一对指令控制 CPU 是否响应中断。用 CLI 关闭中断，用 Sti 开放中断。

二、内存读写

在高级语言和汇编语言程序中，一般是通过符号名字访问内存空间的。编译程序、汇编程序为符号名分配相对地址，操作系统给程序分配空间，程序装载时再将相对地址转换为内存地址(静态重定位)，或者在程序运行时，再将相对地址转换为内存地址(动态重定位)，但是也允许直接使用内存地址读写。

PC 机的内存地址可用段寄存器 CS，DS，SS，ES 的值和偏移量表示。例如内存地址 FFFF0H，分别用 FFFF：0000H，FFF0：00F0H，F000：FFF0H 表示。不同的表示方法分别对应段值 FFFFH、偏移量 0，段值 FFF0H、偏移量 00F0H，段值 F000H、偏移量 FFF0H，在汇编语言中，可向相应的寄存器置值指定内存地址。

在 C 语言中，可以直接将内存地址 0XFFFF0 赋给地址指针，如

```
#include <dos.h>
char far *p;
...
p=0xffff0;
...
```

也可以利用库函数 FP_SEG 与 FP_OFF 求得某地址的段值和偏移量。例如：

```
#include <dos.h>
char far *p;
unsigned int seg_val;
unsigned int off_val;
...
seg_val=fp_seg(p);
off_val=fp_off(p);
```

利用这种办法，可以对某些有专门用途的区域进行读写，比如内存低地址 1K 字节大小的中断向量表，可读 0：0—0：1024 的地址；可以对单色显示器的缓冲区 B0000H 之后 4K 字节，或彩色图形显示缓冲区 B8000H 之后 16K 字节进行直接传送以实现快速显示。

在 BASIC 程序中，下述语句组

```
def seg=&HFFFF
A = 0
```

即赋予 A 的内存地址为 &HFFFF0。

例如，借助 debug，pctool，CV 等调试工具，很容易读出中断向量表，以找到某个中断子程

序的入口地址或判断此表是否已染上某种病毒，利用程序也不难把它读出来。

三、外设口地址操作

1. 系统提供了直接与外设交换信息的指令

(1)汇编语言中用 in,out,格式是

```
in al,DX
```

它从端口 DX 读一个字节到 AL 寄存器。用

```
out DX,AL
```

将 AL 寄存器的内容送往端口 DX。

(2)C 语言中 inp,outp 函数起到上述 in ,out 的作用，格式是

```
#include <conio.h> /* 在说明 inp,outp 的文件前头包含 */
```

函数

```
int inp(port);
unsigned port;
```

的功能是从 Port 所指端口读入一个字节，返回给调用者。

```
int outp(port,value);
unsigned port;
int value;
```

outp 的功能是将 value 的当前值送往端口 port，

(3)在 BASIC 语言中也有类似的函数，它们的名字是 int 和 out，使用格式为

```
inp 地址
out 地址,参数
```

2. 外设口地址约定

PC 机中支持外设工作的芯片主要有三个：

①8259 中断控制器

②8255 可编程的外设接口

③8253 计时器

它们连在总线上，给出了固定的端口编号。表 2.1 列出了每个芯片的地址。

CPU 就是通过这些 I/O 端口读写和控制外设的。注意每个芯片的口地址可能不止一个。每个这样的芯片都是从事专门工作的处理机，例如 8253 能够计算或计时由软件或硬件发出信号的事件，也可用来当作时钟报时用。每个芯片都有一些内部寄存器，我们可以用 out 指令写往适当的端口给寄存器置值，以控制芯片的工作，这就是可编程的意思。同样也可以通过 in 指令读取寄存器的值以了解它们的状态。

3. 8259 中断控制器

微机工作的时候，各种不同的中断随时都可能发生，例如键盘随时可能有按键操作，磁盘驱动器随时可能有 I/O 完成，多个中断信号可能随机地同时发生。CPU 却只有一根中断信号输入线，即每次只能处理一个中断。8259 就管理各种中断信号，暂存、排队并送一个中断信号给 CPU，可以说 8259 是 CPU 专管中断的秘书。

表 2.1 外设接口芯片的部分 I/O 地址

8259 中断控制器	
20H	中断命令寄存器
21H	中断屏蔽寄存器
8253 计时器	
40H	通道 0 门栓寄存器存取口
41H	通道 1 门栓寄存器存取口
42H	通道 2 门栓寄存器存取口
43H	命令寄存器
8255 可编程外围接口	
60H	输入口"PA"
61H	输入/输出口"PB"
62H	输入口"PC"
63H	8255 命令寄存器(设定为 99H)
3BAH—3DAH	单显,彩显状态寄存器
2F8H—2FFH	异步通信端口
3F8H—3FFH	异步通信端口

8259 管理 8 路中断信号,编号 0—7,它每次送一个中断信号及相应的中断类型码给 CPU,CPU 一方根据不同类型有对应的中断处理程序。8 路中断信号在控制总线上分别称为 IRQ0—IRQ7。

对于同时发生的中断,8259 按优先级排队,0 号中断即时钟中断的优先级最高,并依次降低,7 号为最低。CPU 在处理完每一个中断时,必须及时告诉 8259,可以处理下一个中断,8259 便将下一个中断送给 CPU。表 2.2 列出了中断来源。

表 2.2 中断来源

8259 输入	型态码	设 备
IRQ0	08H	计时器(8253 通道口)
IRQ1	09H	键盘
IRQ2	0AH	彩色图形界面
IRQ3	0BH	未用
IRQ4	0CH	串行口(RS232)
IRQ5	0DH	硬盘
IRQ6	0EH	轮船
IRQ7	0FH	打印机

8259 编程控制包含两个步骤,一是置中断屏蔽寄存器 IMR,以指明 CPU 接收哪些中断,不理睬哪些中断。二是置中断命令寄存器。中断屏蔽寄存器的口地址是 21H,它的每个 bit 对应于一个中断源,该位为 1 时,对应的中断被屏蔽,8259 将不会送中断信号给 CPU,只有为 0 的位所对应的中断才是 CPU 所关心的。例如,假设我们要使除了时钟和键盘以外所有中断被屏蔽,就可用下述方式达到:

```
MOV   AL 0FCH
out   21H,AL
```

CPU 的中断标志，若发了 CLI 命令，置 IF = 0，则所有中断都被屏蔽，无论 8259 的 IMR 怎样设置，CPU 都不会理睬任何中断。

关于中断命令寄存器的编程十分简单，其端口地址 20H，只要送一个信号 20H 给它，表明本次中断已处理完毕，要求 8259 向 CPU 发下一个中断。

```
MOV   AL ,020H
out   20H,AL
```

4. 8259 可编程外围接口

8259 可以支持键盘、扬声器、组态开关等外设的工作。它有三个端口，叫做 PA，PB，PC。对应端口地址 60H，61H，62H。此外还有一个命令寄存器端口 63H，开机时，BIOS 送一个值 99H 到 63H 以启动 8255，初始化使 PA，PC 当作输入口，而 PB 当作输出口，可以设定 PB 的值来选择输入是 PA 还是 PC。

程序 2.1 读 8255 端口，以确定当前系统中配备的磁盘驱动器总数。

当 PB 口高位置 1 时，PA 口最高两位便是我们所要的数值。汇编语言程序如下：

```
; 程序(段)2.1 确定磁盘驱动器数目
IN AL,61H      ;读端口 PB
OR AL,80H      ;
OUT 61H,AL     ;PB 口高位置 1
IN AL ,60      ;读 PA 口
NOT AL
MOV CL,6
SHR AL,CL      ;PA 口右移 6 位，AL 中得到了磁盘驱动器总数
```

该程序应在寄存器 AL 中返回所要的值。

5. 8253 计时器

8253 计时器芯片能执行许多计时和计数功能。其内部有三个编号为 0，1，2 的独立的计数器，每一个“计数器通道”都可以被设置来执行六种不同模式的操作。模式编号为 0～5。三个通道可同时分别执行不同模式的操作。

8253 每个通道有关的硬件都是相同的，每个通道包含一个 16 位的“门闩”寄存器和一个 16 位的“计数器寄存器”，每个通道各有两个输入信号，叫做“时序脉冲”和“闸”，以及一个输出信号。使用 8253 时，通常将一个额定的计数值写到“门闩寄存器”内，它再转给“计数器寄存器“，每次当一个脉冲信号出现在“时序脉冲输入”上时，“计数器寄存器”的值就减 1，当其值减到 0 时，便产生一个信号到输出线上，从而达到计时计数的目的。设定计时器通道的模式将完全决定这些动作如何发生。

8253 芯片的端口地址为 40H，41H，42H，43H，前三个地址对应于三个通道，可通过此地址读写门闩寄存器的内容。43H 是 8253 的命令寄存器。

命令寄存器的最高两位用来选用通道，其余各位则定义这个通道如何操作门闩及怎样和它的门闩寄存器通信。门闩寄存器是 16 位的，端口读写每次却只有 8 位，因此需要两次操作才能完成对门闩寄存器的读写。要存取整个门闩寄存器的话，先置命令寄存器的第 4、5 位为 1，然后连续发两条 IN 或 OUT 指令，读或写门闩寄存器，先是低字节 LSB，后是高字节 MSB。假

若仅仅存取其中一个字节，首先按图中所示，适当置命令寄存器第 4、5 位为所需的值，再发一条 IN 或 OUT 指令即可。命令寄存器的 1—3 位用来设置六种操作模式之一。最低位用来指定数据是二进制格式还是 BCD 格式递减。

无论哪种模式，“时序脉冲输入”上的信号决定了计数器递减的速率，“闸输入”信号根据模式可以用来控制时序脉冲信号能否进入计数器，或者用来指示计数动作开始。任何计数动作的结果均出现在输出线上，这个信号根据模式可能为单一脉冲，固定的电压水平，或者是周期性重复的信号。模式 2 和 3 在提供周期性信号时很有用，模式 3 产生一个对称的方波到输出线上，其频率等于输入的时序脉冲频率除以我们设定给门闩寄存器的计数值。

8253 每个计时器通道都有专门的用途。0 通道用来产生 0 阶中断，8H 类中断；通道 1 的输出则用来周期性地发出数据请求信号给 DMA 以进行内存的刷新；而通道 2 的输出则送往扬声器产生音响效果。通道 2 的输出也可以经由 8255 的输入口，PC(62H)上的第 5 位取到。

每个通道的时序脉冲输入都连接到一个 1.193188MHZ 的信号上，因此每个脉冲的周期为 1/1.19318MHZ＝840NS。通道 0 和 1 的闸输入都连接到一个高电位信号上，一直可以工作。在开机时，BIOS 启动通道 0 工作于第 3 种模式，其计数值为 0000H ，这是最大可能计数值，结果要经过 65536 个计数周期后才再回到 0，因此通道 0 的输出便成为 1.19HZ/6 5536＝18.2HZ 的方波。假定时钟中断未被屏蔽，每秒种便会发生 18.2 次，或者每 55ms 一次。BIOS 便利用这个定时中断来记载时间，放在一个 32 位的内存单元(46CH—46FH)之中，磁盘驱动器马达靠这个时钟来维持正常工作，因此一般来讲，我们不要去改变它。BIOS 将通道 1 设定为在模式 2 方式下工作；计数值采用 18，每 15ms 发一个 DMA 内存刷新脉冲。因此通道 1 的工作千万不要更改，通道 2 则可以随心所欲地使用。

程序 2.2 作为用 8255，8253 芯片端口，从而与 PC 硬件打交道的编程实例，我们来写一个发声程序。

声音是由 8253 的通道 2 输出信号给扬声器产生的。为了 8253 能够选择通道 2 工作，就要发命令给 8253 命令寄存器，其端口地址为 43H。而 8253 的工作又需要先由 8255 选通。

下面我们列出此程序(我们叫它 sound.asm)清单，然后再做些解释。

```
;程序 2.2  发声程序 sound.asm
STACK   SEGMENT PARA STACK 'STACK'
        DB 256 DUP(0)
STACK   ENDS
DATA    SEGMETNT PARA PUBLIC 'DATA'
        FREQ DW 1989
DATA    ENDS
CODE    SEGMENT PARA PUBLIC 'CODE'
START   PROC FAR
        ASSUME CS:CODE
        PUSH DS
        MOV AX,0
        PUSH AX
        MOV AX ,DATA
        MOV DS ,AX
        ASSUME DS:DATA  ;
```

```
;
                IN AL ,61H
                OR AL,3
                OUT 61H,AL        ;
                MOV AL,0B6H
                OUT 43H,AL
                MOV BX,FREQ
                MOV AL,BL
                OUT 42H,AL
                MOV AL ,BH
                OUT 42H,AL
                mov ah,0
                int 16h
                IN AL,61H
                AND AL,0FCH
                OUT 61H,AL
                RET
    START ENDP
    CODE ENDS
    END START
```

在数据段设立了变量 Freq，取值 1989 是因为

$$119318\times 106HZ/1989=600HZ$$

可以让 8253 产生 600Hz 的方波。

代码由三部分组成，第一部分设置 8255 端口地址 PB，以便允许 8253 计时器通道 2 输出信号进到扬声器电路。具体说

```
IN    AL,61H
OR    AL,3
OUT  61H,AL
```

设置端口 PB，打开“扬声器数据”(第 1 位)和“定时器门选通 2”信号。

第二部分对 8253 通道 2 编程，使其产生 600HZ 方波，语句组

```
MOV   AL,0B6H
OUT   43H,AL
```

置 8253 命令寄存器，选通道 2。先读 LSB，再读 MSB，采用二进制方式。接着按指定的频率 Freq，先后向通道 2 门闩寄存器送了频率的低字节和高字节以后，扬声器就发声了。

程序中最后一部分是等待用户敲任意键结束，返回 DOS。如果没有按键，扬声器将持续地蜂鸣。

2.2 访问系统数据结构

一、访问内存低地址的通信区

从 0：400 到 0：5FFH 的 RAM 专用于 BIOS，DOS，以及应用程序的通信区。系统引导时，以及有关 BIOS 调用，DOS 功能调用执行时，将代表系统配置，系统工作状态的数据写入其中特定的单元。系统程序员可直接访问这些区域，以获系统状态的内部信息，或者和其它应用

程序通信。有关资料建议用户不要改变 BIOS、DOS 用的通信数据。

1. 区域分配

(1)400H－4ABH:BIOS 通信区

BIOS 在这个区域中存贮大量参数和状态标志。尽管其中大多数参数可以 通过 BIOS 调用得到,但也有一些需要进行直接内存访问。下面给出此区域的基本结构,重点介绍对程序员有实际意义的若干部分(偏移量均以十六进制计)。

地址偏量	内容
400－407	RS－232 适配器地址
408－40F	并行打印机地址
410－411	设备标志(通常仅在引导系统时设置)
412	初始化标志
413－414	以 K 字节为单位的内存大小
415－416	I/O 通道用的内存缓冲区大小
417	键盘状态字
418	第二个键盘状态字
419	留给备用键盘使用
41A－41B	键盘输入字符循环缓冲区首址
41C－41D	键盘输入字符循环缓冲区末址
41E－43D	键盘输入字符的循环缓冲区
43E－448	软盘数据
449－466	显示器参数,包括显示宽度,光标形状等
467－46B	磁带机参数
46C－470	时钟数据(参阅上节有关 8253 的叙述)
471	Break 标志,若按下 Break 键,此标志为 1
472－473	重置标志。如果此字标志为 1234,则 BIOS 初始化子程序将不对内存作检测,因此一旦复位,便进行热启动
474－477	硬盘数据
478－47F	IBM PC jr 用
480－483	额外的键盘数据区
484－4AB	仅用于 EGA 附加显示器参数

(2)4ACH－4EFH:BIOS 保留

(3)4F0H－4FFH:用户通信区。这个区域可由任何程序自由使用。例如,若干独立但有关联的应用程序可通过此区域交换数据。

(4)500H－5FFH:DOS 通信区

2. 使用实例

程序 2.3 获得可用内存大小。BIOS 通信区的 413H－414H 记载了这个值,可用下述程序得到这个值:

```
; 程序(段) 2.3 读可用内存大小
XOR aX,aX
mov es,aX
```

```
mov aX,eS:(413H)
```

运行后于 AX 中得到以 K 字节为单位的可用内存大小。

程序 2.4 直接从内存读取时钟数据。

```
5 '程序 2.4 读时钟程序 TIMES.BAS
10 CLS
20 DEF SEG=0
30 T=PEEK(&H46C)+PEEK(&H46D)*16^2+PEEK(&H46E)*16^4+PEEK(&H46F)*16^6
40 TOLD=T
50 DEF SEG=0
60 T=PEEK(&H46C)+PEEK(&H46D)*16^2+PEEK(&H46E)*16^4+PEEK(&H46F)*16^6
70 DEF SEG
80 DT=T-TOLD
90 LOCATE 8,20
100 PRINT USING "####.#####";DT/18.2
110 GOTO 50
```

BIOS 数据区 0：46c～0：46f 存放时钟数据，该程序可在屏幕上快速地显示时钟值。

程序 2.5 使系统进行快速热启动。方法是置 0：472H—0：473 H 单元为 1234H，转去执行 FFFF0H 处开始的引导程序。

```
; 程序 2.5 快速热启动程序
;BIOS SEGMENT AT 0FFFFH
REBOOT LABEL FAR
BIOS RNDS
XOR AX,AX
MOV ES,AX
MOV WORD PTR ES:(472H),1234H
JMP REBOOT
```

程序 2.6 下列 BASIC 程序可读系统数据区 0：400H，检查是否配有 RS232 适配器

```
80 ' 程序 2.6 检查是否配有 RS232 适配器
100 Def seg = 0
110 Port = Peek(&H400)+ PeeK(&H401)*16^2
120 if port = O then print "没有配备 com1"
130 Def seg
```

二、访问 PSP 程序段前缀

DOS 将一个可执行程序装入内存时，为准备程序运行，在程序代码之前建立一个 256 个字节的数据块 PSP，内容包括 DOS 用于进程管理的参数，程序运行时要求 DOS 提供的信息，管理文件输入/输出及交换文件数据的区域。PSP 的结构列于表 2.3。其中：字节 128 又用作命令行参数长度单元，字节 129—255 又用于存放命令行参数。

1. 重要的 PSP 域

对于系统程序员来说，PSP 中特别重要的一个子集是：①内存项部地址，②段大小，③DOS 文件表，④环境地址，⑤文件控制块 FCB，⑥命令行，⑦默认的磁盘传输地址。下面我们分别来讨论。

(1)偏移量 02h：内存顶部

此域内容是当前已分配给程序的内存块顶部的地址。如果被 DOS 装载的是 . com 文件，在它装入时分配掉所有可用内存。此时“内存顶部”域实际上包含的是用户内存顶部的地址，在此地址之上已没有可供分配的内存了。如果装载的 . exe 文件，则初始分配的内存量取决于文件头中标明的数值，而不是把装载时所有可用的内存。

表 2.3　程序段前缀(PSP)的结构

字节 0—1	存放 1 条 INT 20H 指令
字节 2—3	内存大小，以节(16 字节)计算
字节 4	保留未用
字节 5—9	DOS 功能调用入口地址(为与 CP/M 兼容而设)
字节 10—13	存放 INT 22H 入口
字节 14—17	存放 INT 23H 入口
字节 18—21	存入 INT 24H 入口
字节 22—23	存放父进程的 PSP 段地址
字节 24	标准输入设备的打开文件表单元(存放 SOFT 号)
字节 25	标准输出设备的打开文件表单元
字节 26	标准错误设备的打开文件表单元
字节 27—43	打开文件表单元
字节 44—45	存放环境段地址
字节 46—49	存放用户栈地址指针
字节 50—79	保留未用
字节 80—81	存放 1 条 INT 21H 指令
字节 82	存放 1 条段间返回指令(RETF)
字节 83—84	保留未用
字节 85—91	扩展 FCB 的头(其中：字节 91 用于属性)
字节 92—107	第一个缺省的 FCB
字节 108—123	第二个缺省的 FCB
字节 124—127	保留未用
字节 128—255	缺省的 DTA

都分配给它。可以用两种方法来确定 . com 文件或 . exe 文件已分配了的及未分配的总的内存容量：

①检查 psp 偏移量 02 处的域以确定已分配给该程序的内存量。由于 . com 文件无更多内存可用，故到此为止。

②对 . exe 文件，因装载后很可能还有空闲空间，可用 DOS 功能调用 int 21h 的 4 8h 子功能(分配内存)来确定以节为单位的空闲内存大小。