第5章 中断技术基础

中断技术既是 CPU 管理、服务计算机系统的有效手段，也是程序设计的一种特殊技术。本章将概要介绍中断技术及 80x86 中断系统的基本概念，并在此基础上讲解中断服务程序的编写及应用技术。

5.1 什么是中断技术

CPU 作为计算机系统的核心部件，系统所进行的任何工作都离不开它，如执行人们提交给它的程序，处理系统内发生的硬、软件故障，为系统中的输入/输出设备提供服务等。从原理上说，CPU 无论做何种工作都是通过执行相应的程序来完成的。

除了执行人所提交的程序外，CPU 所做的其他工作都有明显的不确定性，即不能确定何时需要做（如不能确定故障何时会发生、键盘何时会被按下等）。因此，CPU 在执行一个程序期间，计算机系统中很可能会发生其他需要 CPU 去处理的事件（如发生了故障、键盘被按下等），如果这些事件比 CPU 正在执行的任务更为紧迫，就需要 CPU 暂停正在执行的程序，先去处理这些事件，处理完后，再返回到原来的程序继续执行。显然，这种突发事件把 CPU 正在执行的程序从中打断了，因此，这种工作方式被称为“中断方式”。为实现中断方式而采用的硬、软件技术称为“中断技术”。CPU 对突发事件的处理称为“中断服务”，而为完成中断服务所执行的程序称为“中断服务程序”。

中断技术是硬、软件结合的技术。硬件方面，在 CPU 内部有中断响应电路和中断允许与屏蔽电路，在 CPU 外部有中断控制器以及各个设备接口中的中断请求电路；软件方面，则需要编写各种初始化程序和中断服务程序。

5.2 80x86 中断系统简介

5.2.1 中断源类型

在一个计算机系统中，需要采用中断方式来处理的事件多种多样，通常把这些事件称为中断源。对 80x86 系统而言，中断源有以下几类。

（1）数据的输入/输出。一些中、低速输入/输出设备与 CPU 之间的数据传送（如键盘

输入、打印机输出、系统时钟计时等）通常采用中断方式进行，这类中断源称为 I/O 中断。

（2）硬件故障。CPU 对硬件故障（如内存或 I/O 端口奇偶校验出错、电源故障等）的处理均采用中断方式，这类中断源称为硬件故障中断。

（3）软件故障。软件故障是指有符号数加减运算溢出、除法溢出、访存地址越界、内存分配失败等软件运行故障，而非程序的逻辑性错误。软件故障也采用中断方式处理，这类中断源称为软件故障中断。

（4）软中断指令（INT n）。执行软中断指令可以引发一次中断服务（如调用 DOS 或 BIOS 功能程序等），这类中断源称为指令中断。

以上四类事件中，前两类发生在 CPU 外部的硬件装置上，后两类发生在 CPU 内部所执行的程序上，因此，前两类中断被称为外部中断或硬件中断，而后两类中断则被称为内部中断或软件中断。

对内部中断，由于发生在 CPU 所执行的程序上，CPU 可以直接测知，因此，这类中断一旦发生，CPU 将自动进行相应的中断服务。一个例外是，CPU 不会自动检测有符号数加减运算溢出中断（简称溢出中断，当 OF 标志为 1 时发生），而需要在程序中执行 INTO 指令（一条特殊的软中断指令——溢出中断指令）调用 BIOS 的溢出中断服务程序进行处理（见例 3.57）。之所以如此，是因为加（ADD）、减（SUB）运算指令是有符号数和无符号数共用的，故而不能自动处理溢出中断。

与内部中断不同，外部中断发生在 CPU 外部，必须由发生中断的硬件装置发出信号通知 CPU，CPU 才能获知。外部硬件装置向 CPU 发出的中断通知信号称为中断请求信号。如前所述，外部中断包含 I/O 中断和硬件故障中断两类，这两类中断发出的中断请求信号是不同的。对硬件故障中断而言，由于硬件故障会严重影响计算机系统的正常工作，因此，CPU 一旦接到硬件故障中断请求信号，必须做出响应，无条件地去进行中断服务。所以，硬件故障中断也称为不可屏蔽中断（NMI），即不能不响应的中断。与此不同，I/O 中断请求是输入/输出设备需要与 CPU 进行数据传送的请求（如键盘请求 CPU 接收按键信息，打印机请求 CPU 发送下一个字符过来打印等），对这类中断请求，80x86 CPU 将根据标志寄存器中的 IF 标志（中断标志，见 1.3.1 节）的状态，决定是否做出响应（IF=1，响应；IF=0，不响应）。因此，I/O 中断也称为可屏蔽中断（INTR），即可以不予以响应的中断。

设置 IF 标志使用 STI 指令（IF←1）和 CLI 指令（IF←0）。当 CPU 响应可屏蔽中断请求时，会向中断请求方回应一个中断响应（INTA）信号。

5.2.2　中断号与中断向量表

一个计算机系统中有很多中断源，且不同的中断源需要 CPU 执行不同的中断服务程序来进行处理，因此要求 CPU 能够准确识别所有的中断源。为此，计算机系统给每个中断源都编了一个号，以此区分不同的中断源。中断源的编号称为中断号。

80x86 系统的中断号是一个 8 位二进制编号，可以给 256 个中断源编号。这些编号中，有一些已经固定分配给了特定的中断源，有一些被系统保留下来用于以后的扩充，有一些则提供给用户使用，让用户可以为自己设置的中断源编号。表 5.1 为 80x86 系统的中断号资源分配表。

表 5.1 80x86 系统中断号资源分配表

中 断 号	中 断 源	中 断 号	中 断 源
00H	除法溢出中断（BIOS 中断）	20H	程序终止（DOS 中断）
01H	单步中断（BIOS 中断）	21H	DOS 核心功能程序（DOS 中断）
02H	不可屏蔽中断（NMI，BIOS 中断）	22H～3FH	其他 DOS 中断
03H	断点中断（INT 3 指令，BIOS 中断）	40H～5FH	扩充的 BIOS 中断
04H	溢出中断（INTO 指令，BIOS 中断）	60H～6FH	用户可用
05H	屏幕打印中断（Print Screen，BIOS 中断）	70H～77H	扩充的可屏蔽中断（INTR，BIOS 中断）
06H～07H	保留	78H～7FH	未用
08H～0FH	可屏蔽中断（INTR，BIOS 中断）	80H～EFH	BASIC 使用
10H～1FH	其他 BIOS 中断	F0H～FFH	保留

全部 DOS 中断均为指令中断（通过执行 INT n 指令实现中断服务程序调用）。在 BIOS 中断中，中断号为 00H（除法溢出中断）、01H（单步中断）、02H（不可屏蔽中断）、05H（屏幕打印中断）的中断和所有可屏蔽中断，是由中断系统硬件控制实现中断服务程序调用的，其余的 BIOS 中断也是指令中断。DOS 和 BIOS 的这些指令中断均可在程序中使用（使用实例见 4.5 节）。

用户可以用中断号 60H~6FH 定义自己的中断源，并采用指令中断的方式进行中断服务程序调用。当然，用户要为此编写自己的中断服务程序。

对于表 5.1 中那些系统已经明确定义的中断源，其中断号在整个 80x86 系列计算机中是固定不变的（以此保证软件的兼容性），其中断服务程序也是在系统软件（如 BIOS 和 DOS）中编写好的，并在每次开机后，由系统启动程序自动装入内存的固定区域，以保证在中断发生时，CPU 能够找到并执行这些中断服务程序。但用户自定义的中断源都是临时性的，系统不会记住并固定它的中断号，也不会为其编写中断服务程序，更不会自动将其中断服务程序装入内存固定区域。中断号选择、中断服务程序编写、中断服务程序装入内存并获取其入口地址等一系列工作，都需要用户自己去做。这些工作完成后，用户才能通过执行 INT n 指令调用自己的中断服务程序。一旦关机，用户所做的上述工作全部作废，下次需要的话，又要重新再做一次。

如前所述，中断号是区分不同中断源的唯一标识。那么，中断发生时，CPU 是如何获取其中断号的呢？中断号是由中断源向 CPU 提供的，但不同类型的中断源提供中断号的方式有所不同。除法溢出中断、单步中断、不可屏蔽中断和屏幕打印中断的中断号是由专门的硬件电路提供的；可屏蔽中断是由专门的中断控制器（Intel 8259 芯片）统一管理的，其中断号由中断控制器提供；断点中断和溢出中断通过执行特殊的单字节软中断指令 INT 3（机器码 CCH）和 INTO（机器码 CEH）产生，CPU 只要执行这两条指令，就能固定获取中断号 03H 和 04H；DOS 中断和大部分 BIOS 中断的中断号均通过执行双字节的软中断指令 INT n（机器码 CDxxH）产生，该指令中的操作数 n（机器码中的后一字节 xxH）即为中断号。

CPU 获取中断号后如何找到并执行对应的中断服务程序呢？为了在中断发生时 CPU

能够及时执行其中断服务程序进行处理，系统中所有中断源的中断服务程序都是预先装入内存中的，但各自存放的区域不同，入口地址也不同。中断服务程序的入口地址也称中断向量，CPU 只有获取了所需执行的中断服务程序的中断向量，才能转到中断服务程序去执行。为了便于 CPU 获取中断向量，系统在内存中建立了一个中断向量表，所有中断向量按中断号的顺序依次存于表中，中断号小的存于前（低地址方向），中断号大的存于后（高地址方向）。有了中断向量表，CPU 只要根据中断号查表，就可获得对应的中断向量，并转去执行中断服务程序。

中断向量是一个逻辑地址，包含段地址和偏移地址，长度为 4 字节。80x86 系统的中断号共有 256 个，所以中断向量表要能够存储 256 个中断向量，共需占用 1024 字节（即 1KB）的存储空间。为了确保 CPU 查表成功，中断向量表在内存中的存储位置是固定的，每次开机，系统启动程序都会自动将中断向量表装入内存物理地址为 0~1023 的 1KB 空间中（也就是内存中地址最低的 1KB 空间）。在中断向量表中访问中断向量时，段地址设为 0000H，偏移地址就是中断号乘以 4 的乘积。图 5.1 所示为中断向量表的存储示意图。

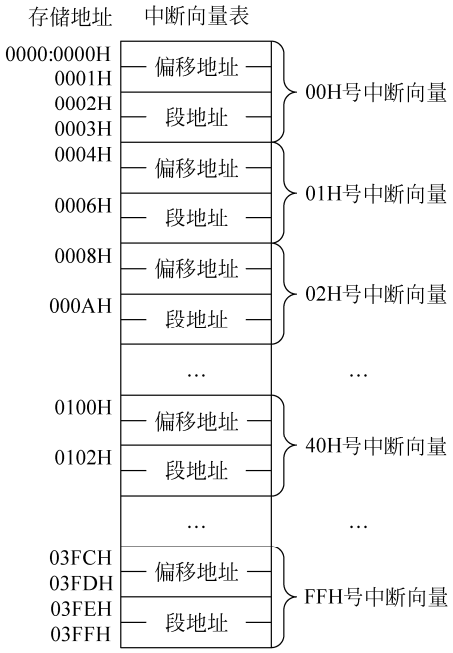

图 5.1　中断向量表存储示意图

CPU 按中断号在中断向量表中找到对应的中断向量后，将其段地址装入 CS，偏移地址装入 IP，即可转入中断服务程序执行。

中断向量表由系统管理，对系统所定义的中断，其中断向量都已固定，并已全部装入中断向量表。中断向量表作为系统的重要内容，被保存在系统磁盘上，每次开机启动时由系统启动程序将其复制到内存最低端。这样，各种系统中断就能得到服务了。

用户如果需要定义自己的中断，可通过在内存中访问中断向量表，向其中写入自己的

中断向量，以实现自己定义的中断服务。当然，用户写入的中断向量并不会被保存到系统磁盘上；也就是说，系统磁盘上的中断向量表是固定不变的，每次使用的只是其复制到内存中的副本，副本可以在内存中修改，但不被保存到系统磁盘上。

5.2.3 中断服务程序及其调用与返回

中断服务程序在形式上被设计成过程（子程序）。与过程相似，中断服务程序也有调用与返回，分别称为中断调用与中断返回。指令中断的服务程序由软中断指令调用执行，其他类型的中断服务程序由中断系统的硬件自动调用执行。为了能够准确地实现中断返回，并保证原来的程序执行环境不被破坏，在中断调用过程中必须先完成标志寄存器和断点地址（即发生中断处的下一条指令的地址）保护（即将标志寄存器和断点处的 CS:IP 入栈），然后才能按中断向量转到中断服务程序执行。中断返回使用的是中断返回指令 IRET，它是中断服务程序的最后一条指令，其作用是从堆栈中弹出断点地址和标志信息，并分别置入 CS:IP 和标志寄存器，从而实现中断返回。

中断服务程序的一般结构框架如下：

```
过程名  PROC  [NEAR/FAR]
  [保护现场]
  [STI]
  过程体   ;中断服务程序主体
  [CLI]
  [恢复现场]
  [发 EOI 命令]  ;仅用于可屏蔽中断服务程序
  IRET  ;中断返回
过程名  ENDP
```

对中断服务程序而言，保护现场是非常重要的，除了指定作为调用参数和返回参数的寄存器外，其他会被中断服务程序修改的通用寄存器和段寄存器（除 CS 外）都要入栈保护。如果允许中断服务程序在执行过程中响应其他可屏蔽中断（I/O 中断）请求，应在保护现场后执行 STI 指令，将 IF 标志置为 1（开中断），因为在中断调用过程中，硬件电路会在转到中断服务程序之前，自动将 IF 清 0。如果在保护现场后执行了 STI 指令，则应该在恢复现场前执行 CLI 指令，将 IF 标志清 0（关中断），以使后面的操作不被其他可屏蔽中断干扰。对可屏蔽中断服务程序，还要在中断返回之前发出中断结束（EOI）命令，以便从硬件上结束本次中断服务。

5.2.4 中断优先级与中断嵌套

有时，系统中可能会有多个中断源同时提出中断请求，但 CPU 一次只能为一个中断源服务，这就需要对提出中断请求的中断源排一个服务顺序。系统中各类中断源的服务顺序是预先定好的，称为中断优先级。设定中断优先级的依据，是各类中断事件的紧迫程度。

80x86 系统的中断优先级如图 5.2 所示。

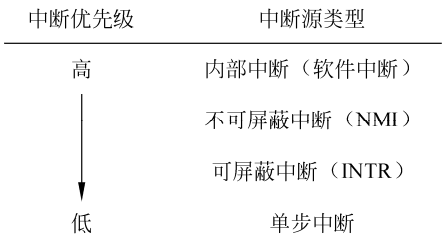

图 5.2　80x86 系统的中断优先级

当 TF 标志（陷阱标志）被设置为 1 时，CPU 每执行完一条指令就会产生一次单步中断（中断号 01H），去执行单步中断服务程序；至于单步中断服务程序的功能，可由使用单步中断的软件系统自行设计。单步中断一般只用于程序的调试，如在 debug.exe 中利用单步中断对程序单步执行，每执行完一条指令，就能对程序执行的所有中间结果进行观察与分析，以此达到程序调试的目的。在所有类型的中断里，单步中断的优先级是最低的。

从表 5.1 中可知，可屏蔽中断本身又包含 8 种（中断号 08H~0FH），它们也有不同的优先级。这 8 种中断的优先级，可通过对 8259 中断控制器编程来进行设置（有多种设置方式）；系统初始化程序是按“中断号越小，优先级越高”的方式设置这 8 种中断的优先级的，这也是系统的默认设置。

有了中断优先级，CPU 就会严格按优先级由高到低的顺序来进行中断服务。

CPU 在执行某个中断服务程序的过程中，又转去执行了另一个中断服务程序，这种现象称为中断嵌套。如果 CPU 在执行某个中断服务程序的过程中，出现了新的内部中断（如遇到了一条软中断指令或出现了除法溢出等），或发生了不可屏蔽中断，则由于这两类中断的不可屏蔽性，中断嵌套会立刻产生。如果 CPU 当前执行的中断服务程序不是可屏蔽中断类的服务程序，且中断服务程序中已开中断（执行了 STI 指令），则当可屏蔽中断发生时，也会产生中断嵌套。比较特殊的是，CPU 在执行一个可屏蔽中断服务程序的过程中，又发生了新的可屏蔽中断，这时，只有在当前执行的服务程序中已开中断，且新发生的中断的优先级高于正在服务的中断的优先级时，中断嵌套才会出现。

中断也可能形成多级嵌套，其嵌套深度没有限制。但由于每次中断处理过程都要利用堆栈进行断点地址及程序执行现场信息的保护，所以，中断嵌套的深度实际上受到堆栈容量的约束。所以，定义堆栈容量时，要充分考虑这个问题。

5.3　如何设置自己的中断服务

从程序设计的角度来看，中断服务程序调用也是一种特殊的程序设计技术，它可以设计出一种与具体程序无关的公用代码，供不同的程序以软中断的方式进行调用。

对于系统已经定义好的中断，其中断服务程序也由系统提供，计算机用户可以直接利用这些资源。例如，可以在自己的程序中直接用软中断指令调用所需的中断服务程序（如调用 DOS 和 BIOS 功能程序），可以直接利用系统提供的各种软、硬件故障中断服务程序

进行故障处理，而无须自己编写这些程序。

如果用户要设置自己的中断服务，则需要完成以下几项工作。

1．选择一个合适的中断号

由表 5.1 可知，60H~6FH 是用户可以使用的中断号。一般情况下，用户都应该在此范围内选择自己的中断号。有时，用户出于某种特殊需要，要改变系统中某个现有的中断服务功能，当然就直接用该中断原来的中断号。

2．编写中断服务程序

编写中断服务程序与编写子程序类似，其结构框架见 5.2.3 节。

对于采用软中断指令调用的中断服务程序，往往要涉及参数传递问题。由于中断服务程序没有固定的宿主程序，因此，在与中断服务程序传递参数时，不能直接通过内存变量名来传递，但可以通过寄存器或堆栈来传递。此外，也可以通过向中断服务程序传递参数地址，以达到传递参数的目的。

需要注意的是，对于堆栈传递参数的情况，由于 IRET 指令不能清除堆栈中的参数（与 RET n 指令不同），所以需要中断服务的调用程序自己来清除。

3．设置中断向量

将中断服务程序的入口地址装入中断向量表。

设中断号为 n，中断服务程序过程名为 INT_SER_PROC，则可用以下指令序列设置中断向量：

```
CLI                              ;关中断
MOV  AX,0000H
MOV  ES,AX                       ;将中断向量表的段地址装入 ES
MOV  BX,4*n                      ;将 n 号中断向量的偏移地址装入 BX
MOV  AX,OFFSET  INT_SER_PROC     ;取中断服务程序入口的偏移地址
MOV  ES:[BX],AX                  ;将中断服务程序入口的偏移地址装入中断向量表
MOV  AX,SEG  INT_SER_PROC        ;取中断服务程序的段地址
MOV  ES:[BX+2],AX                ;将中断服务程序的段地址装入中断向量表
STI                              ;开中断
```

以上指令序列通过直接访问中断向量表的方式设置中断向量。此外，DOS 的 21H 号中断中也有专门的中断向量设置功能：

功能号：25H

调用参数：AL（包含中断号）；

DX（包含中断服务程序入口的偏移地址）；

DS（包含中断服务程序的段地址）。

返回参数：无。

上述中断向量设置过程也可用以下指令序列完成：

```
MOV  DX,OFFSET  INT_SER_PROC     ;中断服务程序入口的偏移地址装入 DX
MOV  AX,SEG  INT_SER_PROC        ;取中断服务程序的段地址
PUSH  DS                         ;保护 DS
MOV  DS,AX                       ;中断服务程序的段地址装入 DS
MOV  AL,n                        ;中断号装入 AL
```

```
MOV  AH,25H                          ;功能号装入 AH
INT  21H                             ;功能调用，完成中断向量设置
POP  DS                              ;恢复 DS
```

如果用户是要改变系统中某个现有的中断服务功能，则需重新编写其中断服务程序，也要重新填写中断向量表，用新的中断向量替换原来的中断向量。但是，用户对现有系统中断服务的修改只是临时性的，当用户的任务执行完后，应该恢复原来的中断向量，使系统的中断服务恢复原状。这就需要在修改中断向量之前，保护原有的中断向量。下面假设 n 为某系统现有中断的中断号，新中断服务程序的过程名为 NEW_INT_PROC，则可用以下指令序列处理中断向量：

```
CLI                                  ;关中断
MOV  AX,0000H
MOV  ES,AX                           ;将中断向量表的段地址装入 ES
MOV  BX,4*n                          ;将 n 号中断向量的偏移地址装入 BX
PUSH  ES:[BX+2]                      ;保护原中断向量中的段地址
PUSH  ES:[BX]                        ;保护原中断向量中的偏移地址
MOV  AX,OFFSET  NEW_INT_ PROC        ;取新中断服务程序入口的偏移地址
MOV  ES:[BX],AX                      ;将新中断服务程序入口的偏移地址装入中断
                                     ;向量表
MOV  AX,SEG  NEW_INT_PROC            ;取新中断服务程序的段地址
MOV  ES:[BX+2],AX                    ;将新中断服务程序的段地址装入中断向量表
STI  ;开中断
```

该指令序列中的第 5 行和第 6 行的 PUSH 指令将原中断向量入栈保护，在用户任务完成后， 再用 POP 指令从堆栈中取出原中断向量，并填写到中断向量表中原来的位置即可。

为避免用户直接访问中断向量表，DOS 的 21H 号中断中也提供了一个取中断向量的功能：

功能号：35H

调用参数：AL（包含中断号）。

返回参数：BX（包含中断向量中的偏移地址）；

　　ES（包含中断向量中的段地址）。

以下指令序列用 21H 功能中的 25H 和 35H 号功能，实现上述保护原中断向量、设置新中断向量的工作：

```
MOV  AL,n                            ;中断号装入 AL
MOV  AH,35H                          ;功能号装入 AH
INT  21H                             ;功能调用，取出原中断向量，并置于 ES:BX 中
PUSH  ES                             ;保护原中断向量中的段地址
PUSH  BX                             ;保护原中断向量中的偏移地址
MOV  DX,OFFSET  NEW_INT_PROC         ;新中断服务程序入口的偏移地址装入 DX
MOV  BX,SEG  NEW_INT_PROC            ;取新中断服务程序的段地址
PUSH  DS                             ;保护 DS
MOV  DS,BX                           ;新中断服务程序的段地址装入 DS
MOV  AH,25H                          ;功能号装入 AH
```

```
INT  21H                          ;功能调用，完成中断向量设置
POP  DS                           ;恢复 DS
```

4．让中断服务程序驻留内存

这项工作并非必需。

通常，用户自定义的中断服务程序是作为一个过程，包含在用户的源程序中的。当用户程序执行结束后，它所占用的内存空间都会被操作系统收回，包括中断服务程序在内的全部程序代码都会作废。如此，这个中断服务程序就只能在包含它的程序范围内使用了。如果想让自己编写的中断服务程序能为其他程序、甚至其他用户所用，就必须使它在宿主程序结束后，仍能驻留内存。

DOS 的 21H 号中断中提供了一个终止当前程序，并将程序驻留内存的功能：

功能号：31H

调用参数：AL（包含一个返回码）；

DX（包含需要保留的内存大小，以节为单位）。

返回参数：无。

AL 中的返回码用来指出程序终止的原因：00H—正常终止；01H—用 Ctrl+C 终止；02H—因严重设备错误而终止；03H—因调用 31H 号功能而终止。描述所需保留的内存大小时，要指出欲保留的节数（1 节等于 16 字节）。

正确计算出欲保留的内存节数，是确保程序驻留成功的关键。每个程序在内存中，都有一个 256 字节（相当于 16 节）的程序段前缀（PSP，其内容可参阅相关资料）作为其头部，且 31H 号功能在保留内存时，是从 PSP 开始的，因此，在计算保留内存的节数时，必须计入 PSP 的 16 节。因为系统在为程序分配内存空间时，是按各个段定义的顺序依次分配的，所以，如果在欲保留的指令代码前，还定义有堆栈（段）和数据（段），也必须分别把它们折算成节数，并计入总节数中。指令代码部分要保留的内存，至少是从代码段起始处开始，到欲保留的代码全部包含在内为止的一块空间。当然，也可以将整个代码段全部保留。

【例 5.1】 将例 4.11 中的 STRTODEC 和 DECTOSTR 这两个过程改造成中断服务程序 INT_STRTODEC 和 INT_DECTOSTR，并驻留内存。

分析：首先，要在 60H~6FH 范围内为 INT_STRTODEC 和 INT_DECTOSTR 选择中断号，在此不妨选择 6AH 和 6BH；其次，要为两个中断服务程序设计好参数传递方式。考虑到堆栈传递参数不是很适合中断服务程序，在此都采用寄存器进行参数传递。具体参数传递方案如下。

（1）INT_STRTODEC。

中断号：6AH

调用参数：DS:SI（包含待转换的数字串的串首地址，串长不超过 10）；

CX（包含待转换的数字串的串长）；

DS:DI（包含转换结果的存储地址，一个双字单元的地址）。

返回参数：无。

（2）INT_DECTOSTR。

中断号：6BH

调用参数：DS:SI（包含待转换的数的存储地址，一个双字单元的地址）；
　　DS:DI（包含存放转换结果的存储区首地址）。
返回参数：无。
存放转换结果的存储区结构为

```
SUM  DB  ?,10 DUP(0)
```

其中，SUM 的首字节用于存放所转换的十进制数的实际位数，其后的 10 字节存储区用于存放转换所得的十进制数字串，且最低位数字存于最高地址端（即地址为 SUM+10 的单元）。

下面编写程序，对上述两个中断服务程序进行设计，并设置其中断向量，最后将其驻留内存。

```
SSEG  SEGMENT  STACK          ;定义堆栈段
  STLBL  DW  32  DUP(?)
  SLEN=($-STLBL+15)/16        ;SLEN 为堆栈段的节数
SSEG  ENDS
CSEG  SEGMENT                ;定义代码段
ASSUME  CS:CSEG,SS:SSEG
INT_STRTODEC  PROC           ;定义中断服务程序 INT_STRTODEC
        PUSH  AX             ;保护现场
        PUSH  BX
        PUSH  DX
        PUSH  BP
        STI                  ;开中断
        XOR  AX,AX
        MOV  [DI],AX         ;置存放转换结果的存储单元初值为 0
        MOV  [DI+2],AX
  CONT: MOV  BX,10           ;下面将十进制数字串转换为对应的数值
        MUL  BX              ;将上一步转换结果的低位部分乘以 10
        PUSH  DX             ;保存到堆栈
        PUSH  AX
        MOV  AX,[DI+2]       ;取出上一步转换结果的高位部分
        MUL  BX              ;将上一步转换结果的高位部分乘以 10
        MOV  DX,AX           ;因转换结果只有 32 位，如超出，只能舍弃
        XOR  AX,AX
        POP  BX
        POP  BP
        ADD  AX,BX           ;将高、低两部分乘以 10 之后的结果相加→DX:AX
        ADC  DX,BP
        MOV  BL,[SI]         ;从数字串中取出一位数
        SUB  BL,30H          ;将其转换为对应数值
        XOR  BH,BH
        ADD  AX,BX           ;将该位数与前面求得的结果相加→DX:AX，完成一个数位的转换
        ADC  DX,0
        MOV  [DI+2],DX       ;保存转换结果的高位部分，低位部分将直接投入下一步转换
```

```
        INC  SI              ;修改数字串指针，指向下一位数字
        LOOP  CONT           ;循环控制，进入下一步转换
        MOV  [DI],AX         ;保存最终转换结果的低位部分
        CLI                  ;关中断
        POP  BP              ;恢复现场
        POP  DX
        POP  BX
        POP  AX
        IRET                 ;中断返回
INT_STRTODEC  ENDP           ;中断服务程序 INT_STRTODEC 定义结束
;
INT_DECTOSTR  PROC           ;定义中断服务程序 INT_DECTOSTR
        PUSH  AX             ;保护现场
        PUSH  DX
        PUSH  BX
        PUSH  CX
        STI                  ;开中断
        PUSH  DI             ;保护结果存储区首地址，后面还要使用
        ADD  DI,10           ;使 DI 指向存放结果数字串最低位的单元
        MOV  AX,[SI]         ;将待转换数的低位部分存入 AX
        MOV  DX,[SI+2]       ;将待转换数的高位部分存入 DX
        MOV  BX,10           ;BX 作为除数，转换采用“除 10 取余”法
        XOR  CL,CL           ;CL 清零，作为数字位数计数器
BEGIN:  CMP  AX,0            ;判断转换结束条件
        JNE  CONV
        CMP  DX,0
        JE  OVER             ;若需转换之数为 0，则结束转换
CONV:   XOR  DX,DX           ;进入转换
        DIV  BX              ;被除数低位部分除以 10
        PUSH  DX             ;结果入栈保存
        PUSH  AX
        MOV  AX, [SI+2]      ;取出被除数高位部分
        XOR  DX,DX
        DIV  BX              ;被除数高位部分除以 10
        MOV  [SI+2],AX       ;保存商的高位部分
        POP  AX
        MOV  [SI],AX         ;保存商的低位部分
        POP  AX              ;AX 中为余数的低位部分
        CMP  DX,0            ;DX 中为余数的高位部分
        JE  GETONE       ;若余数高位部分为 0，则余数低位部分即为本次除 10 所得余数
        DIV  BX              ;余数高位部分不为 0，则余数部分继续除以 10
        ADD  [SI],AX         ;修改商的值
        ADC  WORD PTR [SI+2],0
        MOV  AX,DX           ;将最终的余数存入 AX
GETONE: ADD  AL,30H          ;将余数（即转换所得的一位十进制数）转换为数字符
```

```
        MOV  [DI],AL          ;将数字符存入结果存储区中恰当的位置
        DEC  DI               ;调整 DI 指针
        INC  CL               ;数字位数计数
         MOV  AX, [SI]        ;取出新的被除数（即上一次除 10 所得的商）
        MOV  DX, [SI+2]
        JMP  BEGIN            ;转去继续实施转换
OVER:  POP  DI                ;从堆栈中取出结果存储区首地址
        MOV  [DI],CL          ;将数字位数存入指定位置
        CLI                   ;关中断
        POP  CX               ;恢复现场
        POP  BX
        POP  DX
        POP  AX
        IRET                  ;中断返回
INT_DECTOSTR  ENDP            ;中断服务程序 INT_DECTOSTR 定义结束
;
CLEN=($-INT_STRTODEC+15)/16   ;CLEN 为欲保留的代码（含以上两个中断服务程序）的节数
;
;下面是主程序部分，完成中断向量设置，并实现程序驻留。主程序部分无须驻留内存
START:  MOV  DX,OFFSET  INT_STRTODEC  ;设置 INT_STRTODEC 的中断向量
        MOV  AX,SEG  INT_STRTODEC
        MOV  DS,AX
        MOV  AL,6AH           ;中断号 6AH
        MOV  AH,25H           ;25H 号功能，设置中断向量
        INT  21H              ;功能调用，完成中断向量设置
;
        MOV  DX,OFFSET  INT_DECTOSTR  ;设置 INT_DECTOSTR 的中断向量
        MOV  AX,SEG  INT_DECTOSTR
        MOV  DS,AX
        MOV  AL,6BH           ;中断号 6BH
        MOV  AH,25H           ;25H 号功能，设置中断向量
        INT  21H              ;功能调用，完成中断向量设置
;下面完成程序驻留
        MOV  AH,31H           ;功能号 31H 装入 AH
        MOV  AL,00H           ;返回码装入 AL
        MOV  DX,SLEN+CLEN+16  ;需保留内存的节数装入 DX，16 为 PSP 的节数
        INT  21H               ;功能调用，终止程序，并完成驻留
CSEG  ENDS                    ;代码段结束
END  START                    ;源程序结束
```

以上程序完成两个中断服务程序的中断向量设置并驻留内存后，其他程序就可以通过执行软中断指令来调用这两个中断服务程序。

【例 5.2】 在例 5.1 已将 INT_STRTODEC 和 INT_DECTOSTR 驻留内存的基础上，请编写程序，实现例 4.11 所要求的功能。

分析：因例 4.11 所要求的两种转换功能已由驻留内存的两个中断服务程序来完成，所

以本例只需编写一个程序（类似于例 4.11 中的主程序）来调用这两个中断服务程序即可。

程序编写如下：

```
SSEG  SEGMENT  STACK          ;定义堆栈段
  DW  32  DUP(?)
SSEG  ENDS
DSEG  SEGMENT  ;定义数据段
  DECSTR1  DB  "32798455"     ;任意定义十进制数字串 DECSTR1
  N1=$-DECSTR1                ;N1 为数字串 DECSTR1 的串长（即数字位数）
  DECSTR2  DB  "1955782"      ;任意定义十进制数字串 DECSTR2
  N2=$-DECSTR2                ;N2 为数字串 DECSTR2 的串长（即数字位数）
  DEC1  DD  ?                 ;DEC1 用于存放 DECSTR1 所对应的数值
  DEC2  DD  ?                 ;DEC2 用于存放 DECSTR2 所对应的数值
  BUF  DD  ?                  ;BUF 用于在计算过程中暂存数据
  SUM  DB ?,10 DUP(0) ;SUM 的首字节用于存放和的十进制数字位数，其后的 10 字节存储区
                      ;用于存放两数之和所对应的十进制数字串
DSEG  ENDS
CSEG  SEGMENT  ;定义主程序（调用程序）所在代码段
ASSUME  CS:CSEG,DS:DSEG,SS:SSEG
START: MOV  AX,DSEG
       MOV  DS,AX
       LEA  SI,DECSTR1  ;用 SI 传递 DECSTR1 的首地址
       MOV  CX,N1       ;用 CX 传递 DECSTR1 的串长
       LEA  DI,DEC1     ;用 DI 传递转换结果的存储地址
       INT  6AH         ;调用 6AH 号中断服务程序，将十进制数字串 DECSTR1 转换为数值
       LEA  SI,DECSTR2  ;用 SI 传递 DECSTR2 的首地址
       MOV  CX,N2       ;用 CX 传递 DECSTR2 的串长
       LEA  DI,DEC2     ;用 DI 传递转换结果的存储地址
       INT  6AH         ;调用 6AH 号中断服务程序，将十进制数字串 DECSTR2 转换为数值
       MOV  AX,WORD PTR DEC1  ;DEC1→DX:AX
       MOV  DX,WORD PTR DEC1+2
       ADD  AX, WORD PTR DEC2  ;DEC1+DEC2→DX:AX
       ADC  DX, WORD PTR DEC2+2
       MOV  WORD PTR BUF,AX  ;DX:AX→BUF
       MOV  WORD PTR BUF+2,DX
       LEA  SI,BUF      ;用 SI 传递 BUF 的首地址
       LEA  DI,SUM      ;用 DI 传递 SUM 的首地址
       INT  6BH         ;调用 6BH 号中断服务程序，将数值转换为十进制数字串并存入 SUM
       MOV  AH,4CH
       INT  21H
CSEG  ENDS
END  START
```

需要注意的是，只有在例 5.1 中的程序执行完后未重新启动系统的情况下，例 5.2 中

的程序才能顺利实现 6AH 号和 6BH 号中断服务程序的调用。如果在执行完例 5.1 中的程序后重新启动了系统，则前面所做的 6AH 号和 6BH 号中断向量设置和中断服务程序驻留均失效，在这种情况下，例 5.2 中的程序不可能得到正常地执行。

从表 5.1 中可知，中断号 08H~0FH 分配给了可屏蔽中断。其中，08H 号中断称为日时钟中断，该中断由系统中的定时/计数器以 18.2Hz（即每秒 18.2 次）的频率产生，其中断服务所完成的工作，是以 24 小时为周期，为系统提供计时信息。在 08H 号中断服务程序中，包含了一条 INT 1CH 指令，调用了 1CH 号中断服务程序，因此，1CH 号中断的产生频率也是 18.2Hz。但是，系统提供的 1CH 号中断服务程序仅包含了一条 IRET 指令，并无任何实质性的服务功能。实际上，1CH 号中断是系统提供给用户自行开发使用的一个中断，用户可以编写自己的 1CH 号中断服务程序，来完成一些具有定时要求的周期性的工作。

【例 5.3】 要求在不影响系统正常工作的前提下，在屏幕的最下面一行，自右向左不断周期性移动显示字符串″THIS IS MY COMPUTER″。

分析：可从以下几个方面确定设计的方向。

（1）由于要求周期性移动显示，因此需要周期性的时间控制。如前所述，采用 1CH 号中断服务，是实现稳定的时间控制的便捷途径。

（2）由于既要周期性移动显示字符串，又不能影响系统的正常工作，因此，周期性移动显示字符串工作不能独占 CPU。也就是说，不能让 CPU 专门单独运行周期性移动显示字符串的程序。因为那样的话，在结束字符串显示程序之前，CPU 就无法去执行系统中的其他任务。解决这个矛盾的有效方法，就是利用 1CH 中断来移动显示字符串，并将中断服务程序驻留内存。这样，定时未到时不会执行中断服务程序，系统可以去做其他工作，定时一到就产生中断，去调用 1CH 中断服务程序进行一次字符串的移动显示，显示完后，又返回到系统的正常工作中。1CH 中断每 1/18.2 秒产生一次，用户可自行设置计数参数来调整移动显示的速度。

（3）由于屏幕最下面一行（第 24 行）用于移动显示字符串，所以系统其他工作只能使用屏幕第 0~23 行。一旦光标移到第 24 行，就要将屏幕第 0~23 行向上滚动一行，而第 24 行不受影响。

（4）移动显示字符串时，从屏幕右端开始，每次向左移动一个字符位置，直到字符串最后一个字符移出屏幕左端，然后又从屏幕右端开始，逐渐向左移动显示，如此周期性循环显示下去。移动显示过程中，要特别注意字符串从右端移入和从左端移出这两个特殊阶段，因为在这两个段中，字符串只是部分显示的。此外，还要注意清除移动尾迹。

（5）由于本例需要比较精准的显示控制，所以应该采用 BIOS 的 10H 号中断功能（见 4.5.3 节）来完成各种显示操作。

程序编写如下：

```
SSEG  SEGMENT  STACK          ;定义堆栈段
  STLBL  DW  32  DUP(?)
  SLEN=($-STLBL+15)/16        ;SLEN 为堆栈段的节数
SSEG  ENDS
CSEG  SEGMENT
ASSUME  CS:CSEG,SS:SSEG
```

```
INT_1CH  PROC                ;定义新的 1CH 号中断服务程序
      PUSH  AX               ;保护现场
      PUSH  BX
      PUSH  CX
      PUSH  DX
      PUSH  BP
      PUSH  ES
      MOV  AX,CS
      MOV  ES,AX
      MOV  AH,0FH            ;功能号 0FH→AH，准备读取当前显示参数
      INT  10H               ;调用功能，读取当前显示参数
      MOV  CS:[PAGENO],BH    ;保存当前显示页号
      MOV  AH,03H            ;功能号 03H→AH，准备读取当前页的光标位置
      INT  10H               ;调用功能，读取当前页的光标位置
      CMP  DH,23             ;当前光标所在行号与 23 比较
      JNA  DISP              ;若未超过 23，转 DISP
      MOV  AH,06H            ;若超过 23，功能号 06H→AH，准备将第 0~23 行向上滚动一行
      MOV  AL,1              ;设置滚动行数为 1
      MOV  CH,0              ;设置滚动区左上角行号为 0
      MOV  CL,0              ;设置滚动区左上角列号为 0
      MOV  DH,23             ;设置滚动区右下角行号为 23
      MOV  DL,79             ;设置滚动区右下角列号为 79
      MOV  BH,07H            ;设置新滚入行（即新的第 23 行）的显示属性为 07H（即黑底白字）
      INT  10H               ;调用功能，实现滚屏
      MOV  AH,02H            ;功能号 02H→AH，准备设置光标位置
      MOV  BH,CS:[PAGENO]    ;取当前显示页号→BH
      MOV  DH,23             ;设置光标所在行号为 23
      MOV  DL,0              ;设置光标所在列号为 0
      INT  10H               ;调用功能，完成光标位置设置，光标位置(23,0)
 DISP: DEC  CS:[TIMER]       ;定时计数器 TIMER 减 1
      JNZ  EXIT              ;若未计到 0，本次不做移动显示，转到 EXIT
      MOV  CS:[TIMER],2 ;若已计到 0，则将 TIMER 恢复为初值 2，并准备进行一次移动显示
      MOV  AH,13H            ;功能号 13H→AH，准备进行一次字符串显示
      MOV  BH,CS:[PAGENO]    ;当前显示页号→BH
      MOV  BL,07H            ;移动显示字符串的显示属性→BL，为黑底白字
      MOV  DH,24             ;显示所在行号 24→DH
      MOV  AL,0              ;字符串显示方式 0→AL
      MOV  DL,CS:[HEAD]      ;本次显示的串首位置（列号）→DL
      CMP  DL,0              ;判断串首是否已经移出屏幕左边界（第 0 列）
      JL  NSH  ;若串首已移出屏幕左边界，则转 NSH，计算实际应从哪个字符开始显示
      LEA  BP,CS:[STR]       ;否则，将实际串首的偏移地址→BP，段地址已在 ES 中
      MOV  CX,SLEN           ;需显示的字符串原始长度→CX
      CMP  CS:[TAIL],78      ;判断本次显示的串尾位置是否已进入屏幕右边界（第 78 列）
      JLE  SHOW              ;若串尾已进入右边界，则转 SHOW，按串的原始长度完整显示串
      ADD  CX,78             ;否则，计算需显示的实际串长
```

```
        SUB  CL,CS:[TAIL]
        SBB  CH,0
        JMP  SHOW               ;转 SHOW，按求得的实际串长显示串
  NSH:  LEA  CX,CS:[STR]        ;下面计算本次应从串的第几个字符开始显示
        NEG  DL
        ADD  CL,DL
        ADC  CH,0
        MOV  BP,CX              ;将开始字符的偏移地址→BP，段地址已在 ES 中
        XOR  CH,CH              ;下面计算本次实际需显示的串长
        MOV  DL,CH
        MOV  CL,CS:[TAIL]
        INC  CX                 ;实际需显示的串长→CX
SHOW:   INT  10H                ;调用功能，完成本次字符串显示
        DEC  CS:[HEAD]          ;一次显示完成后，修改串首和串尾位置，为下一次显示做准备
        DEC  CS:[TAIL]
        CMP  CS:[TAIL],0        ;判断串尾是否已移出屏幕左边界
        JGE  EXIT               ;若尚未移出，则转 EXIT，准备结束本次操作
        MOV  CS:[HEAD],78   ;若串尾已移出左边界，则重新初始化串首和串尾位置，准备下一轮
        MOV  CS:[TAIL],78+SLEN-1
  EXIT: ;此处，新 1CH 号中断服务完成，准备调用原 1CH 号中断服务程序，兼顾其他用户的需求
        ;下面为原 1CH 号中断服务程序的返回，将标志寄存器和返回点地址入栈
        PUSHF                   ;标志寄存器内容入栈
        PUSH  CS                ;返回点的段地址（即当前 CS）入栈
        LEA  AX,RETPT           ;取返回点的偏移地址
        PUSH  AX                ;返回点的偏移地址入栈
        JMP  DWORD  PTR  CS:[OLD_1CH_IP]  ;用段间转移方式转入原 1CH 号中断服务程序
RETPT:  POP  ES                 ;恢复现场
        POP  BP
        POP  DX
        POP  CX
        POP  BX
        POP  AX
        IRET  ;中断返回
;
;下面定义 1CH 中断服务程序所需的变量
PAGENO  DB  ?           ;用于保存当前显示页号
HEAD  DB  78            ;串首位置（列号），初始值 78，为屏幕右边界位置
STR  DB  'THIS IS MY COMPUTER',20H  ;需显示的字符串，以空格符（20H）结束
SLEN=$-STR              ;SLEN 为串长
TAIL  DB  78+SLEN-1 ;串尾位置（列号），初始值 78+SLEN-1，在屏幕右边界之外
TIMER  DB  2            ;定时计数器，初始值 2，控制每隔 2/18.2 秒移动显示一次
OLD_1CH_IP  DW  ?   ;用于保存原来的 1CH 号中断向量的偏移地址
OLD_1CH_CS  DW  ?   ;用于保存原来的 1CH 号中断向量的段地址
;
INT_1CH  ENDP           ;1CH 号中断服务程序定义结束
```

```
;
CLEN=($-INT_1CH+15)/16       ;需驻留内存的代码节数
;
;下面是主程序部分，保存原 1CH 号中断向量，设置新 1CH 号中断向量，并完成驻留
START: MOV  AH,35H
       MOV  AL,1CH
       INT  21H
       MOV  CS:[OLD_1CH_IP],BX
       MOV  CS:[OLD_1CH_CS],ES
;
       MOV  DX,OFFSET  INT_1CH
       MOV  AX,SEG  INT_1CH
       MOV  DS,AX
       MOV  AL,1CH
       MOV  AH,25H
       INT  21H
;
      MOV  AH,31H
      MOV  AL,00H
      MOV  DX,STLEN+CLEN+16
      INT  21H
CSEG  ENDS
END  START
```

本程序执行后，1CH 号中断服务程序被驻留内存，且每隔 1/18.2 秒被自动调用一次。

在以上程序的设计中，采用了一些非常规的设计方法，对此做一些说明。

（1）在 INT_1CH 过程体中定义内存变量。由于 INT_1CH 过程是作为中断服务程序使用的，它不隶属于任何其他程序，因此，中断服务程序内部使用的内存变量不能依赖外部定义，而应包含在服务程序内部，自成一体。由于这种变量与代码放在一起，所以在访问时要用 CS 作为段前缀。

（2）保留并调用了原 1CH 号中断服务。由于 1CH 号中断是系统提供给用户自行开发使用的一个中断，因此，在本程序定义新的 1CH 号中断服务之前，系统的其他用户可能也定义了自己的 1CH 号中断服务。为了避免新的 1CH 号中断服务影响系统其他用户的正常工作，本程序保留了原 1CH 号中断向量，并在新的 1CH 号中断服务完成后，利用已保留的原 1CH 号中断向量，通过特殊的段间转移方式调用了原 1CH 号中断服务程序，这样就满足了系统其他用户对 1CH 号中断服务的要求。

此外，本程序把移动显示区的右边界定在了屏幕第 78 列，而不是屏幕真正的右边界第 79 列。这是为了避免造成屏幕显示混乱。虽然，10H 号中断的 13H 号功能在显示完字符串后，会将光标恢复到显示前的位置，但在显示过程中，光标还是会跟随显示过程向右移动的。如果将屏幕右边界定在第 79 列，那么，在第 79 列显示一个字符后，光标就会自动移到下一行。由于本例中的字符串是在屏幕最下面一行（第 24 行）上显示的，光标下移实际上会造成屏幕自动向上滚动一行（光标仍在第 24 行上），从而将所显示的字符串滚动

到屏幕的其他行，造成屏幕显示混乱。

定义显示用字符串 STR 时，串尾增加的空格符（ASCII 码 20H）是用来清除移动显示时产生的尾迹的。字符串每次向左移动一个字符位置，串尾的空格符正好覆盖掉上一次显示的最后一个有效字符，从而达到了清除移动尾迹的效果。

习题

1．设计一个求两个无符号字节数据最小公倍数的中断服务程序，并驻留内存。

2．设计一个求两个无符号字节数据最大公约数的中断服务程序，并驻留内存。

3．设计一个求百分比的中断服务程序（结果以字符串形式表示，如″30.12%″），并驻留内存。

4．要求在不影响系统正常工作的前提下，在屏幕的右下角位置，动态显示用机时间，格式如下，要求每秒钟更新一次。

```
hh:mm:ss
```