汇编语言程序设计

实验三 计时器设计

2005010130 侯杰

一、实验要求

1．设计一个计时器，模拟计时过程（按键开始计时，按键结束计时）

2．计时尽量精确

3．提供给其他程序进行执行时间计时的功能

二、实验分析

此次实验与以往实验不同，多了与时间相关的处理。考虑到汇编语言课程上所讲的内容，可采用重写系统中断的方式进行处理。

系统执行时每 55ms 就要调用一次中断类型号为 1ch 的中断处理程序，只是系统默认的此中断处理程序并不做任何操作。现在我们重写此中断处理程序，就可以使系统每 55ms 调用一次此程序。在此程序中只要进行适当的计数工作，同时输出到屏幕，就可以观察到数据随时间的变化，继而实现我们的计时器。

三、实现思路

1．实现基本计时器功能

我们重写中断号为 1ch 的中断向量处理程序。在此程序中，我们对数据进行更新。由于系统大约 1 秒钟调用 18 次此程序，所以可以每 18 次调用就更新一次数据，这样就可以看到数据以秒为时间单位进行变化。

我们用定义几个变量 如hour,minute,second来存储时钟各部分的值。开始时各值均为 0.需要更新时，首先是 second 先增 1，然后判断是否满 60，

如果满的话则清零，同时 minute 增 1。同样 minute 判断是否满 60，如果满的话清零，同时 hour 增 1。最后判断 hour 是否满 24，若满的话清零。

每更新一次，都需要将当前的各位值输出。由于内存中是二进制值，所以需要先把二进制转化成十进制，然后变成 ASCII 码，再通过调用 int21h 调用 DOS 功能将其输出。在时钟不同部分之间额外输出“：”作为分隔。同时为保证其时钟的显示效果，每次输出应位于同一位置。所以输出完一次都要回行首。输出控制符 0dh 即可实现。

由于要求摁一次键盘开始计时，再摁一次停止计时，所以主程序中应加入键盘监听部分。这个可以通过监听 60h 号端口键盘输入寄存器来解决。通过判断此端口寄存器最高位的值就可以得知当前是否摁下键盘。

具体实现应注意般中断程序的处理规则，即主过程完成：

（1）监测键盘摁键，有摁键继续执行，否则等待。

（2）备份系统原来中断向量

（3）设置新的中断向量

（4）监测键盘摁键，有摁键继续执行，否则等待。（在等待之时中断程序不停被调用）

（5）恢复原来的中断向量

中断程序完成：

（1）内存中数据更新

（2）内存中数据显示输出

2.实现更精确的计时

由于系统条件限制，最小的计时单位是 55ms，所以采用这种中断计时方式的最小计时单位也是 55ms。在原来设计的基础上，可以再增加毫秒位（millisecond）来增加精确度。毫秒位每次中断调用都进行更新，每次自加 55。当达到 1000 或以上时，millisecond 自减 1000，同时 second 增 1。毫秒位的显示设置为 3 位，和其他位输出采取类似的方式，只不过其他位都只限 2 位输出，毫秒位是 3 位输出。

3．提供给其他程序进行执行时间计时的功能

这是个很有挑战性的题目。在之前我们已经实现了计时，其实计的就是两次键盘摁键之间的系统循环的时间。类似的，如果我们在某个程序运行初始之时为其设置好上文定义的中断，在程序运行完毕之时恢复原中断，这样就可以在程序执行之时，在后台一直运行我们的中断计时程序，从而记录并显示出程序的运行时间。

更进一步，为考虑到方便与其他程序的结合，可以修改原来的计时器程序。将其键盘监听，设置中断，恢复中断与中断程序各自写成独自的过程。当独立运行计时器程序时，利用键盘监听过程控制其余过程的启动。当需要测其他程序的运行时间时，就在程序首尾分别调用设置中断，恢复中断过程，完成计时。这样就等于把计时器程序的除了键盘监听部分的其他部分做成一个库，设置中断（过程 START）以及恢复中断（过程 STOP）作为库的接口，可在其他程序中调用。所以这样一来只要将任意一个程序和库文件 link 在一起，通过调用 START 和 STOP 就可以完成计时并同步显示。

四、具体实现

依照以上思路，编写出计时器程序 TimePro.asm 如下：

（另附 TimePro.asm）

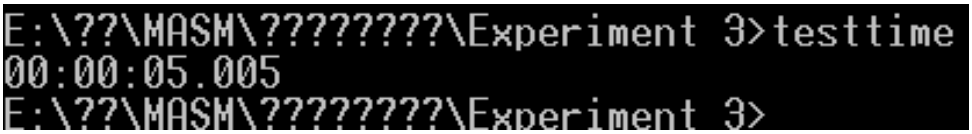
其中 TIMING 为中断处理程序，START 为设置中断的子程序，STOP 为恢复中断的子程序。

编译链接后独立执行可作为时钟显示。摁任意键开始计时，再摁一下计时停止，程序退出。运行情况如下图：

编译后，与需要测定运行时间的程序的.obj 文件通过 link 程序链接在一起（注意此程序在运行开始时调用 START 过程，结束时调用 STOP 过程，而且需要声明 START，STOP 为 extrn 标号），再执行程序，就可以看到程序执行的时间被显示出来。这里我写了一个很简单的小测试程序，里面只有循环语句，目的是拖延时间，然后与 TimePro 组合查看其运行时间。此测试程序内容如下：

（另附 TestTime.asm）

编译成 TestTime.obj 并和 TimePro.obj 链接后运行，结果如下图：



可见 TestTime 程序的运行时间被显示出来。

五、实验中遇到的问题及处理

在实验中遇到一些重要的问题如下：

（1）关于 ds 寄存器内容的管理

由于我的中断程序中用到了数据段 data_time，而当其他程序运行时 ds

指向的并不是这个数据段的地址。为了中断程序在后台运行，每次执行中断程序时都要先将 ds 赋成 data_time 的地址，否则时钟的各位信息就无法从内存中正确的位置取。执行完中断，回到主程序时，同样还要将 ds 的数据恢复成主程序所用的数据段的地址，以便主程序的继续执行。

（2）关于保护现场的重要性

给其他程序计时时，涉及到不少 call 操作。这时保护好现场，即保护好当前寄存器的内容是很重要的。我的 START，STOP 以及中断程序 TIMING 都在执行的开始时将用到的寄存器内容压栈，执行完后再恢复给原寄存器。我在调试时曾忽略这个问题，导致调用这个计时库内的过程后就无法继续执行源程序，或者出现异常并死机。

六、实验后体会与收获

首先，通过本次实验，我锻炼了写中断程序的能力，也学会了处理一些常见的使用问题。实验后，我对中断的理解更深了，同时试用了书写库文件，为第三方程序使用的思想，学会了 public，extrn 等伪操作的使用。

同时，我也不得不承认，这种用中断计时并不是一个好方法。首先它的精确度很差，最小单位才 55ms，远不能精确计时。其次，这样计时需要每 55ms 就进行一次数据更新并显示操作，若是在其他程序运行时作为后台运行，势必会影响主程序的执行效率。我曾观察过，这个程序执行时 cpu 占用率还是很高的。说明其效率确实不佳。

在 C 语言里面 ,常见的程序计时一般是用程序执行后的系统时间减去程序执行前的系统时间，这样在程序执行过程中可以不占用额外资源，效率更高些。所以我希望以后用汇编写计时程序时 ,能先多学习一些获取系统时间方面的功能命

令，不再采用此方法编写计时程序。