实验四实验报告

目录

一、 【个人信息】

院系：数据科学与计算机学院
专业：计算机科学与技术（超算方向）
年级：2016
班级：教务 2 班
姓名：劳马东
学号：16337113
邮箱：laomd@mail2.sysu.edu.cn

二、 【实验题目】

1. 操作系统工作期间，利用时钟中断，在屏幕最边缘处动态画框，第一次用字母 A，第二次画用字母 B，如此类推，还可加上变色闪耀等效果。适当控制显示速度，以方便观察效果。
2. 编写键盘中断响应程序，原有的你设计的用户程序运行时，键盘事件会做出有事反应：当键盘有按键时，屏幕适当位置显示“OUCH! OUCH!”。
3. 在内核中，对 34 号、35 号、36 号和 37 号中断编写中断服务程序，分别在屏幕 1/4 区域内显示一些个性化信息。再编写一个汇编语言的程序，作为用户程序，利用 int 34、int 35、int 36 和 int 37 产生中断调用你这 4 个服务程序。
4. 扩充系统调用，实现三项以上新的功能，并编写一个测试所有系统调用功能的用户程序。

三、 【实验目的和要求】

1. 掌握 pc 微机的实模式硬件中断系统原理和中断服务程序设计方法，实现对时钟、键盘/鼠标等硬件中断的简单服务处理程序编程和调试，让你的原型操作系统在运行以前已有的用户程序时，能对异步事件正确捕捉和响应。
2. 掌握操作系统的系统调用原理，实现原型操作系统中的系统调用框架，提供若干简单功能的系统调用。
3. 学习握掌 c 语言库的设计方法，为自己的原型操作系统配套一个 c 程序开发环境，实现用自建的 c 语言开发简单的输入/输出的用户程序，展示封装的系统调用。

四、 【实验方案】

(一) 实验原理

1. 中断

中断(interrupt)是指对处理器正常处理过程的打断。中断与异常一样，都是在程序执行过程中的强制性转移，转移到相应的处理程序。

1) 硬中断（外部中断）——由外部（主要是外设[即 I/O 设备]）的请求引起的中断
   - 时钟中断（计时器产生，等间隔执行特定功能）
   - I/O 中断（I/O 控制器产生，通知操作完成或错误条件）

■ 硬件故障中断（故障产生，如掉电或内存奇偶校验错误）

2) 软中断（内部中断）——由指令的执行引起的中断

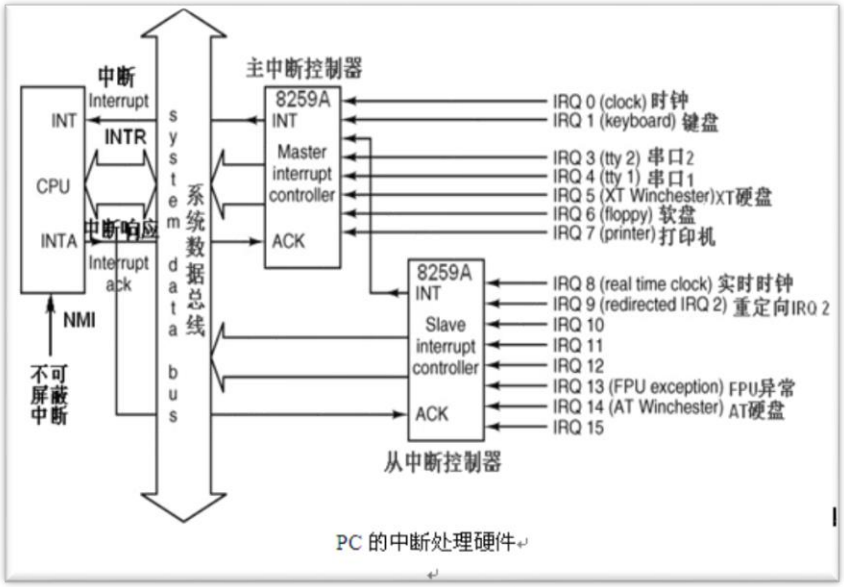
■ 中断指令（软中断 int n、溢出中断 into、中断返回 iret、单步中断 TF=1）

3) 异常/程序中断（指令执行结果产生，如溢出、除 0、非法指令、越界）

2. 中断向量

1) x86 计算机在启动时会自动进入实模式状态

A. 系统的 BIOS 初始化 8259A 的各中断线的类型。



PC 的中断处理硬件

图 1 PC 的中断处理硬件

主/从	中断请求	中断类型
主8259A	IRQ0	Intel 8253/8254可编程间隔计时器，即系统计时器
	IRQ1	Intel 8042键盘控制器
	IRQ2	级联从8259A
	IRQ3	8250 UART串口COM2和COM4
	IRQ4	8250 UART串口COM1和COM3
	IRQ5	在PC/XT中为硬盘控制器 在PC/AT以后为Intel 8255并行端口LPT2
	IRQ6	Intel 8272A软盘控制器
	IRQ7	Intel 8255并行端口LPT1/伪中断
从8259A	IRQ8	RTC（Real-Time Clock，实时时钟）
	IRQ9	无公共的指派
	IRQ10	无公共的指派
	IRQ11	无公共的指派
	IRQ12	Intel 8042 PS/2鼠标控制器
	IRQ13	数学协处理器
	IRQ14	硬盘控制器1
	IRQ15	硬盘控制器2

图 2 主/从 8259A 的初始中断请求类型

B. 在内存的低位区（地址为 0~1023[3FFH]，1KB）创建含 256 个中断向量的表 IVT（每个向量[地址]占 4 个字节，格式为：16 位段值:16 位偏移值）。

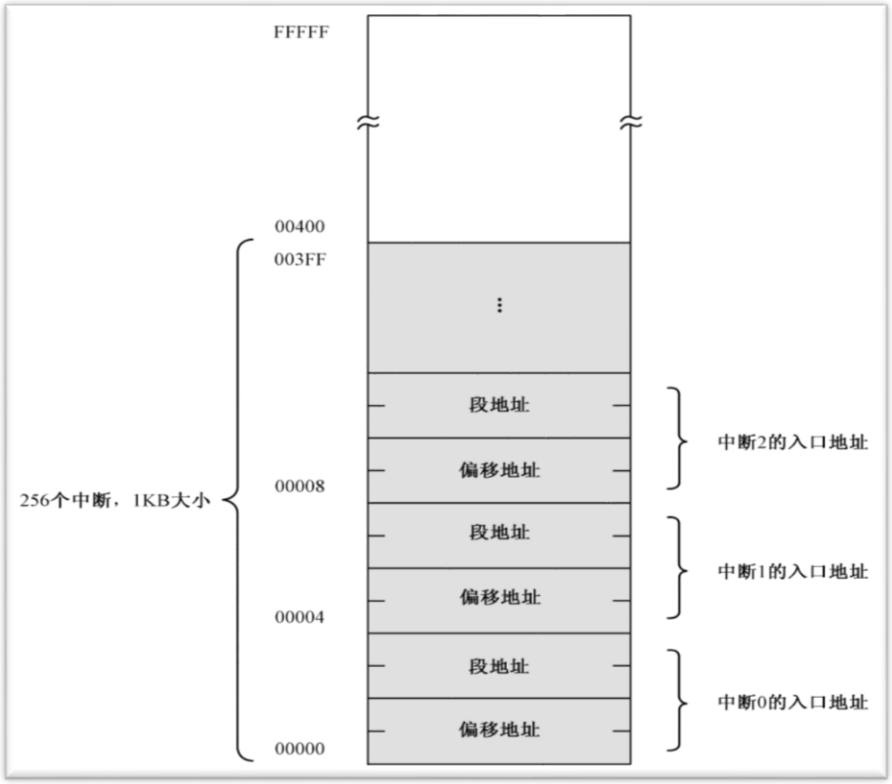


图 3 中断向量地址排列

2) 当系统进入保护模式

A. IVT（Interrupt Vector Table，中断向量表）会失效

B. 需改用 IDT（Interrupt Descriptor Table，中断描述表），必须自己编程来定义 8259A 的各个软中断类型号和对应的处理程序

3. PC 中断的处理过程

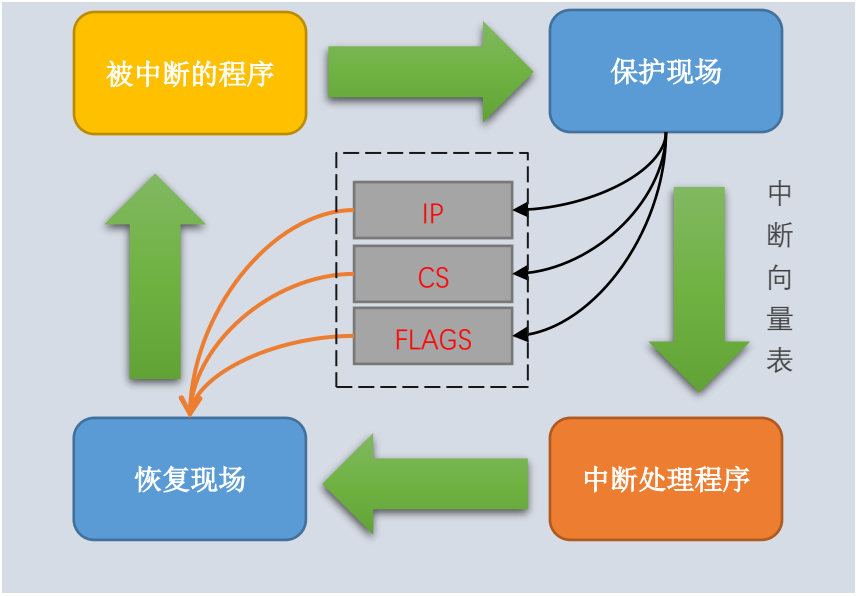


图 4 中断处理过程处理器控制权转移

1) 保护断点的现场
   A. 要将标志寄存器 FLAGS 压栈，然后清除它的 IF 位和 TF 位
   B. 再将当前的代码段寄存器 CS 和指令指针寄存器 IP 压栈
2) 执行中断处理程序
   A. 由于处理器已经拿到了中断号，它将该号码乘以 4（每个中断在中断向量表中占 4 字节），就得到了该中断入口点在中断向量表中的偏移地址
   B. 从表中依次取出中断程序的偏移地址和段地址，并分别传送到 IP 和 CS，自然地，处理器就开始执行中断处理程序了。
   C. 注意，由于 IF 标志被清除，在中断处理过程中，处理器将不再响应硬件中断。如果希望更高优先级的中断嵌套，可以在编写中断处理程序时，适时用 sti 指令开放中断。
3) 返回到断点接着执行
   所有中断处理程序的最后一条指令必须是中断返回指令 iret。这将导致处理器依次从堆栈中弹出（恢复）IP、CS 和 FLAGS 的原始内容，于是转到主程序接着执行。

4. 软中断实现系统服务

它与内核子程序软中断调用方式原理一样，每一种服务由一个子程序实现，指定一个中断号对应这个服务，入口地址放在中断向量表中，中断号固定并且公布给用户，用户编程时才可以中断调用，参数传递可以使用栈、内在单元或寄存器。系统服务分 BIOS 调用和系统调用：

1) BIOS 调用
   BIOS 是固化在只读存储器 ROM 中的一系列输入/输出服务程序，它存放于内存的高地址区域内，除负责处理系统中的全部内部中断外，还提供对主要 I/O 接口的控制功能，如键盘、显示器、磁盘、打印、日期与时间等。BIOS 采用模块化结构，每个功能模块的入口地址都存于中断向量表中。对这些中断调用是通过软中断指令 INT n 来实现的，中断指令中的操作数 n 即为中断类型码。
2) 系统调用
   因为操作系统要提供的服务更多，服务子程序数量太多，但中断向量有限，因此，实际做法是专门指定一个中断号对应服务处理程序总入口，然后再将服务程序所有服务用功能号区分，并作为一个参数从用户中传递过来，服务程序再进行分支，进入相应的功能实现子程序。这种方案至少要求向用户公开一个中断号和参数表，即所谓的系统调用手册，供用户使用。如果用户所用的开发语言是汇编语言，可以直接使用软中断调用。如果使用高级语言，则要用库过程封装调用的参数传递和软中断等指令汇编代码。规定系统调用服务的中断号是 21h。

调用方法：

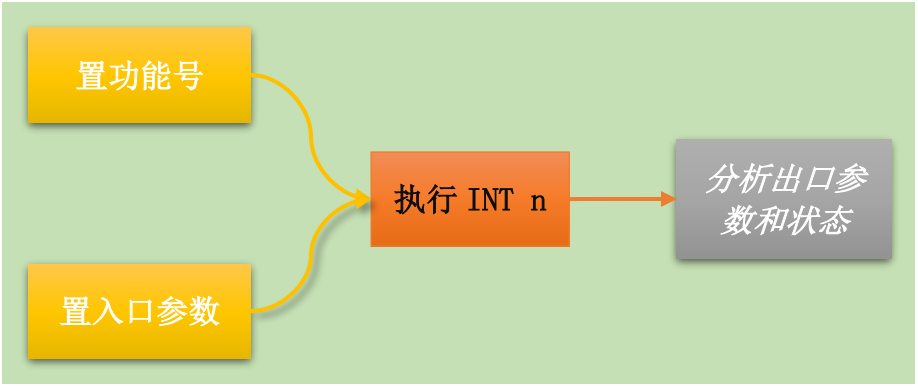


图 5 软中断调用方法

五、 【实验过程】

(一) 时钟中断

1. 时钟中断产生物理原因理解

   OS 时钟的物理产生原因是可编程定时/计数器产生的输出脉冲，这个脉冲送入 CPU，就可以引发一个中断请求信号。在固定的时间间隔都发生一次中断，也是说每秒发生该中断的频率都是固定的，DOS 默认为每秒 16.6666 次。

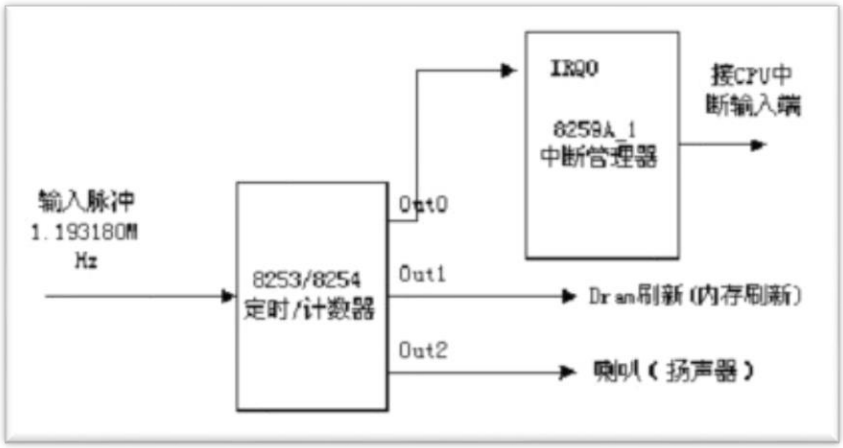


图 6 时钟中断的产生

2. 时钟中断安装

```
; 设置时钟中断向量（08h），初始化段寄存器
    xor ax,ax           ; AX = 0
    mov es,ax           ; ES = 0
    mov word [es:20h],Timer ; 设置时钟中断向量的偏移地址
    mov ax,cs
    mov word [es:22h],ax        ; 设置时钟中断向量的段地址=CS
    mov ds,ax           ; DS = CS
    mov es,ax           ; ES = CS
```

时钟中断处理逻辑

08h*4=20h
IP 低 CS 高

图 7 时钟中断安装代码

3. 中断退出

```
iret:
pop ip
pop cs
pop FLAGS
```

```
end:
    mov al,20h          ; AL = EOI
    out 20h,al          ; 发送EOI到主8529A
    out 0A0h,al         ; 发送EOI到从8529A
    iret
```

图 8 中断返回代码

EOI 是外部中断的中断结束命令，发送 EOI 到主 8529A 和从 8259A，通知 8259A 芯片一个中断完成，8259a 将负责把 ISR 中的位清除，以便以后可以继续接受中断。如果不加的话，8259a 永远收不到中断结束命令，那么就认为某一个中断一直在执行，所以如果遇到比这个阻塞的中断级别低或者相等的中断发生时就不会再响应了。

4. 时钟中断处理逻辑的更改

旧版：从‘!’起，在屏幕（12,39）位置依次输出后面的字符

```
; 时钟中断处理程序
    delay equ 4      ; 计时器延迟计数
    count db delay       ; 计时器计数变量，初值=delay
Timer:
    dec byte [count]          ; 递减计数变量
    jnz end              ;>0：跳转
    inc byte [gs:((80*12+39)*2)]     ; =0：递增显示字符的ASCII码值
    mov byte[count],delay        ; 重置计数变量=初值delay
```

每 4 次时钟中断递增一次

图 9 旧版 Timer

新版：在屏幕最边缘处动态画框，第一次用字母 A，第二次画用字母 B，如此类推；动态变色闪耀，字符存在几秒后消失

```
void cTimer()
{
    if(!flag)   init();
    printCharAt(row1, col1, 'A' + index, (row1 - index + 1) * MAX_ROW + col1);
    printCharAt(row2, col2, ' ', 0x07);

    Move(&row1, &col1, &h1, &v1);
    Move(&row2, &col2, &h2, &v2);

    if (row1 == 0 && col1 == 0)
    {
        index = (index + 1) % 26;
    }
}
```

画完一圈，改变字符

在屏幕特定位置带颜色打印字符。（行列号、字符、颜色）

图 10 新版 Timer

Move 函数从（0,0）位置开始移动字符，在转角处改变方向；字符颜色是行列号和字符的函数，以此达到动态变色和闪耀的效果；通过在字符后面一定距离尾随打印默认颜色的空格，实现字符消失效果。

5. 效果图

根据当前位置判断移动方向并移动(行列号、横竖方向)

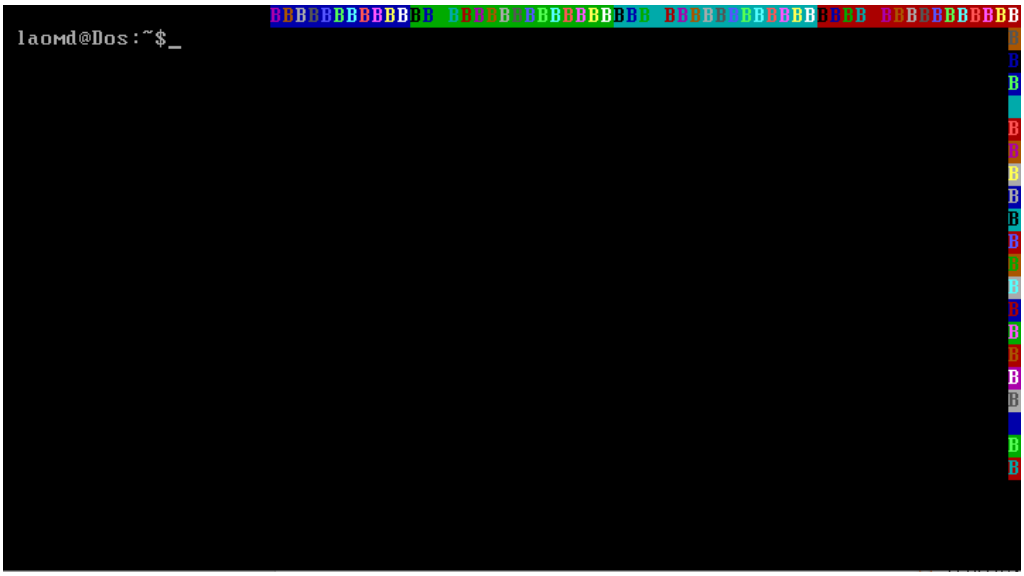


图 11 新版时钟中断效果图

（二） 键盘中断

1. 键盘中断理解

   PC 机中使用的是编码键盘，在键盘内部有一单片机对整个键盘上的字符键、功能键、控制键和组合键进行管理，当从键盘上输入一个键时，键盘上的处理器首先向微机产生中断请求（IRQ1），然后将该键的扫描码传送给主机。而 PC 主机在 IRQ1 的作用下，调用 INT 09 硬件中断读入键盘的扫描码，并转为对应 ASCII 码，存入键盘缓冲区。

2. 键盘中断安装与恢复

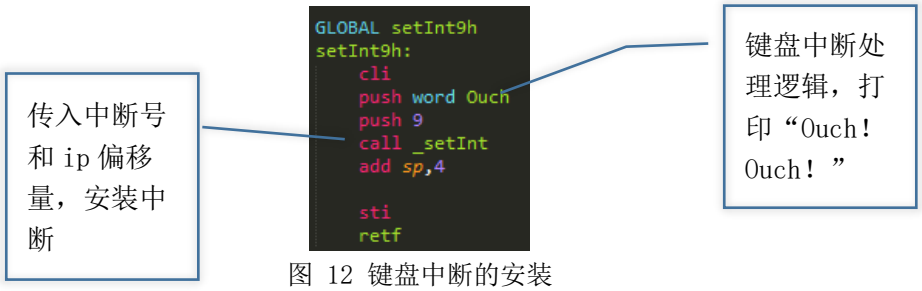


图 12 键盘中断的安装

该函数在操作系统 C 模块中调用，当用户程序准备执行时，操作系统安装键盘中断，恢复时钟中断；当用户程序执行完成后，操作系统恢复键盘中断，安装时钟中断，并继续之前暂停的画框，而不是重新开始。

```
void RunProgram(const Program* program)
{
    int length = numSectors(program->sector.bytes);
    Load(program->sector.cylinder, program->sector.track, program->sector.index, length, program->sector.load_addr);
    resetInt(8);
    setInt9h();
    // cls();
    Run(program->sector.load_addr);
    // cls();
    resetInt(9);
    setInt8h();
}
```

图 13 操作系统控制中断的安装与恢复

除中断向量表中的起始地址不同外，所有中断安装、恢复、退出的逻辑相同，于是把相关代码写进函数_setInt 和 resetInt 和 intEnd 中：

_setInt(int *interr_num*, int *ip_addr*);

Precondition：中断号 *interr_num*，偏移地址 *ip_addr*；

Postcondition：保存并安装中断；

```
; setInt(int interr_num, int ip_addr);
_setInt:
    push ax
    push bx
    push bp
        mov bp,sp
        add bp,2*3
        mov bx,[bp+2]
        mov ax,[bp+4]

        ;保存原来的中断
        shl bx,2
        push bx
        call saveInt      ;ip
        add sp,2
        add bx,2
        push bx
        call saveInt      ;cs
        add sp,2

        ; 设置新中断
        mov bp,bx
        mov word[bp],cs
        sub bp,2
        mov word[bp],ax
    pop bp
    pop bx
    pop ax
    ret
```

保存中断的初始 CS 和 IP 到 int_vector 中，以便不需要中断时恢复

图 14 安装中断代码

resetInt(int *interr_num*);

Precondition：中断号 *interr_num*

Postcondition：恢复中断 *interr_num* 的初始 cs 和 ip

```
; void resetInt(int which)
GLOBAL resetInt
resetInt:
    push ax
    push bx
    push es
    push gs
    push bp
        mov bp,sp
        add bp,5*2

        mov ax,[bp+4]
        mov bx,4
        mul bx
        mov bx,ax

        xor ax,ax
        mov gs,ax
        mov bp,int_vector
        add bp,bx
        mov word ax,[bp]      ;ip
        mov word [gs:bx],ax
        add bx,2
        add bp,2
        mov word ax,[bp]      ;cs
        mov word [gs:bx],ax

        mov byte[color],0
    pop bp
    pop gs
    pop es
    pop bx
    pop ax
    retf
```

键盘中断 0uch 字体颜色置 0（看不见的颜色），解决键盘中断第一次进入时显示 0uch 的问题

从 int_vector 取出中断的初始 cs 和 ip 并放入内存的中断向量表

图 15 恢复中断代码

3. 打印 Ouch

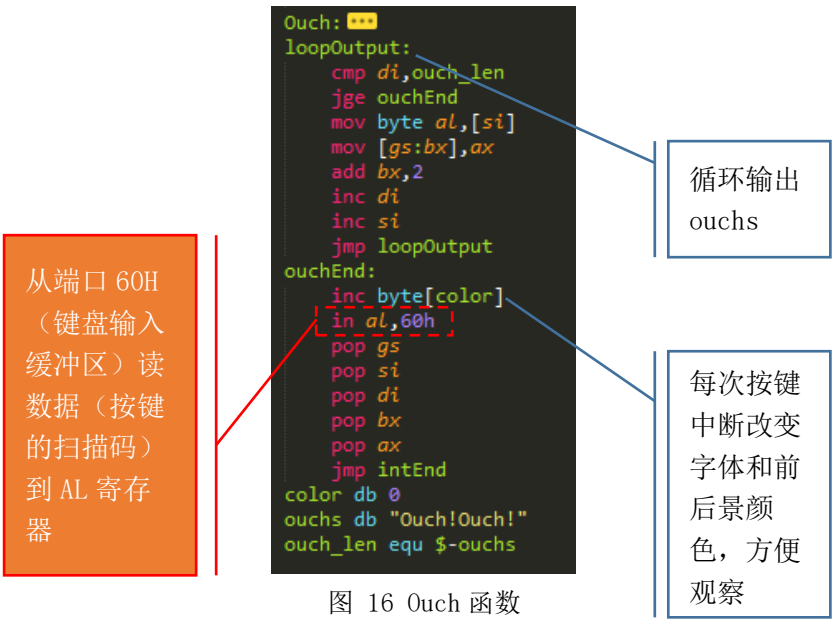


图 16 Ouch 函数

4. 效果图

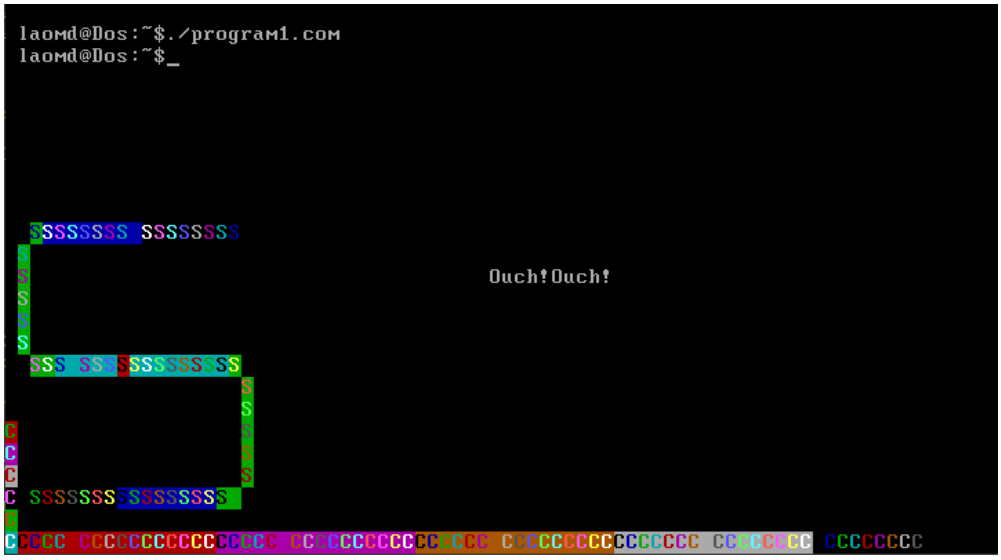


图 17 键盘中断 Ouch 效果图

用“./”命令运行用户程序，上图左下角“S”字样为用户程序的输出结果。用户程序运行期间，时钟中断暂停，每次按键产生键盘中断，在屏幕中间部分显示“Ouch! Ouch!”，每次按键 ouch 的颜色不同。

5. 问题与解决方案

1) 键盘中断只执行了一次，之后按键不再有反应？

这是因为每次按键之后，对应键被存到键盘缓冲区中，由于没有读掉缓冲区，缓冲区没有复位，一直等待被读掉， 于是就出现了程序执行一次键盘中断之后卡住的现象。只要在中断返回前用“in al,60h”指令从 60h 端口（对应键盘输入端口）读缓冲区，就能复位键盘缓冲区，使其可以重新获取按键。

2) 一次按键执行了几次中断？

两次，因为按键按下和抬起均会引起中断。通过修改按键中断处理逻辑来

验证：

```
Ouch:...
; loopOutput:
;       cmp di,ouch_len
;       jge ouchEnd
;       mov byte al,[si]
;       mov [gs:bx],ax
;       add bx,2
;       inc di
;       inc si
;       jmp loopOutput
    mov ah,0eh
    mov al,'A'
    mov bl,0
    int 10h
ouchEnd:
    ; inc byte[color]
    in al,60h
```

以电传方式在当前光标处输出字符‘A’

图 18 验证按键中断次数代码

结果如下图：

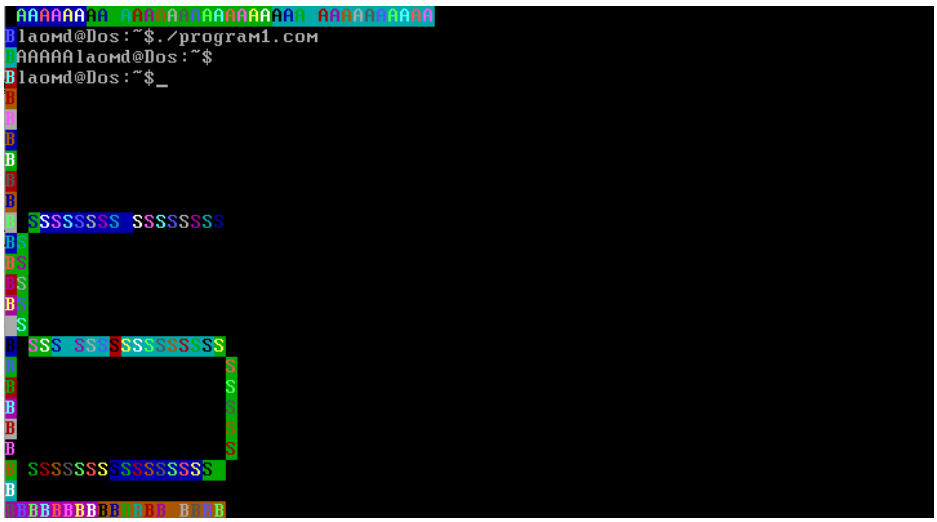


图 19 验证按键中断次数结果图

屏幕左上角输出了 5 个‘A’，第一个‘A’是抬起引发，后面 4 个‘A’是按键两次的输出。由此可见，一次按键对应两次键盘中断。于是，当在命令行打“./”命令回车之后，用户程序运行起来，但出现一个奇怪的现象：还没按键盘就输出“Ouch”了！这是因为在命令行敲回车是按下的动作，此时程序解释命令，发现要运行用户程序，于是安装键盘中断，这是抬起的动作还没发生，当抬起时，键盘中断响应，打印“Ouch”。为了解决这个问题，执行把第一次输出“Ouch”时的颜色置为 0，就看不见输出“Ouch”（实际上输出了），效果上相当于没有输出。

6. 从 Ouch 到音乐“键盘”

音乐“键盘”与打印 Ouch 最大的不同之处是需要知道用户到底按下了哪个键。从上面的分析知道，“in al,60h”读到了键盘缓冲区输入的扫描码，这是获取用户按键的重要方法。但问题又来了，只有扫描码，怎么对应到音乐“键盘”设定的按键以便选择频率？具体讨论请参考后面的【技术点创新点】。

（三） 软中断（22h-25h）

1. 22h 中断安装

```
GLOBAL setInt22h
setInt22h:
    cli
    push word @22h
    push 22h
    call _setInt
    add sp,4

    sti
    retf
```

```
void main()
{
    setInt8h();
    setInt21h();
    setInt22h();
    setInt23h();
    setInt24h();
    setInt25h();
```

时钟中断

系统服务中断

22h-25h 中断

图 20 22h 中断安装函数和操作系统添加软中断代码

同时钟中断和键盘中断，22h 号中断调用_setInt 函数安装，由操作系统启动时添加。@22h 函数是 22h 号中断的处理逻辑，功能是在屏幕左上的 1/4 区域打印“This is 22h!”。

```
@22h:
    cli
    push ax
    push bx
    push di
    push si
    push gs
    mov ax,0xb800
    mov gs,ax

    mov bx,(80*4+4)*2
    mov ah,22
    mov di,0
    mov si,@22hmsg
@22h1:
    cmp di,@22hmsglen
    jge @22hEnd
    mov byte al,[si]
    mov [gs:bx],ax
    add bx,2
    inc di
    inc si
    jmp @22h1
@22hEnd:...
@22hmsg db "This is 22h!"
@22hmsglen equ $-@22hmsg
```

屏幕左上角（4,4）位置，22 号颜色

输出@22hmsg 字符串

图 21 22h 处理逻辑

2. 测试调用 22h 到 25h

23h-25h 安装函数大同小异，只是输出信息、颜色和位置不同而已。测试代码如下：

```
org 1000h
int 22h
int 23h
int 24h
int 25h
```

连续调用 22h 到 25h，如果中间某个中断有误都会没法返回操作系统

图 22 22h-25h 测试代码

测试结果：

图 23 22h-25h 测试结果

22h-25h 分别在屏幕左上、右上、左下、右下 1/4 区域打印关于自己的信息。之所以设计得这么简单，一是考虑到这些中断在以后的实验中会做一些与系统相关的操作，如 DOS 的 22h 用于结束退出当前程序，23h 为 CTRL+BREAK 退出，24h 为出错退出，因此没必要在这个实验中设计过于复杂的功能，只需检查其调用的正确性即可，这才是设计中断的关键；二是 22h-25h 并不是本次实验的重点，它们只是为 21h 的出场做铺垫。需要注意的是，在这个测试程序中，键盘中断仍然有效，只是因为字体颜色与屏幕相同看不到而已。

(四) 21h 系统服务

1. 21h 中断安装

```
GLOBAL setInt21h
setInt21h:
    cli
    push word @21h
    push 21h
    call _setInt
    add sp,4

    sti
    retf
```

图 24 21h 系统功能中断安装

2. 添加具有不同功能的分支

如何通过一个中断来实现多个功能？最简单的做法是指定一个功能号，在中断处理逻辑中判断功能号，进入不同功能的入口。因此，只需要给出功能号（ah）及其对应功能的表格，就能与用户程序间根据约定调用功能。

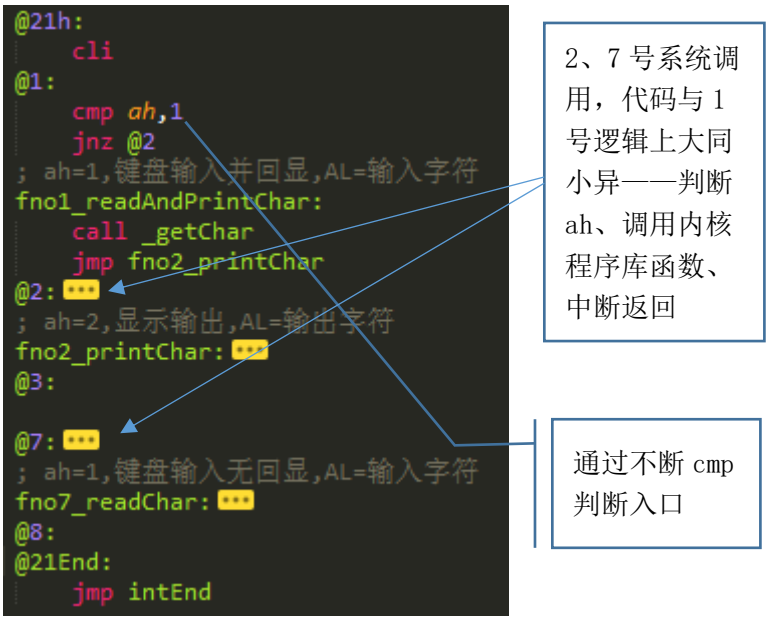


图 25 21h 功能分支

功能号	功能	入口参数	出口参数
1	键盘输入并回显	无	AL=输入字符
2	当前光标处显示字符	AL=输出字符	无
7	键盘输入无回显	无	AL=输入字符

表格 1 系统功能表

3. 内核程序库封装

读取字符：调用 BIOS 的 16H 中断读键盘，读到的字符在 AL 中

```
_getChar:
    xor ax,ax
    int 16h
    ret
```

图 26 内核库——读取字符

输出字符：调用 BIOS 的 10H 中断显示字符，并实现“\r\n”连续输出

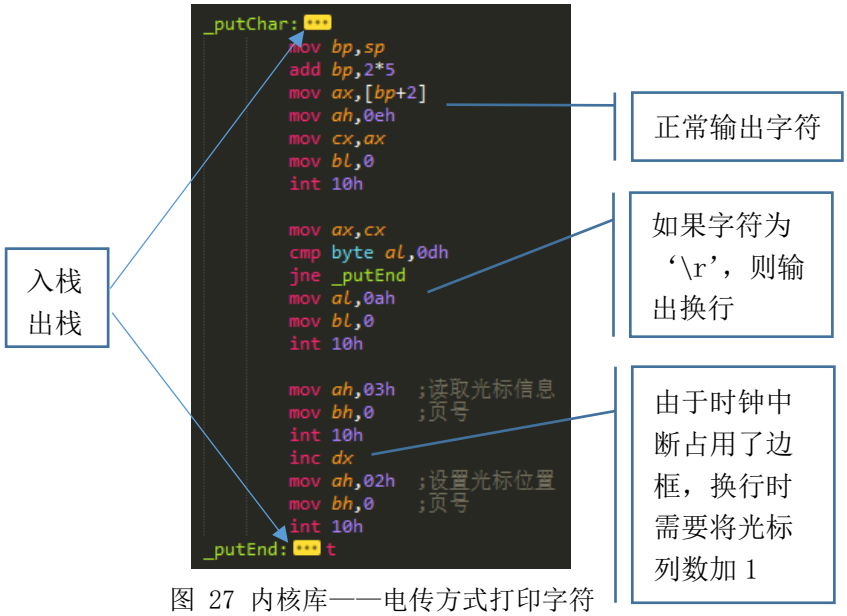


图 27 内核库——电传方式打印字符

4. 非内核程序库改写

旧版：

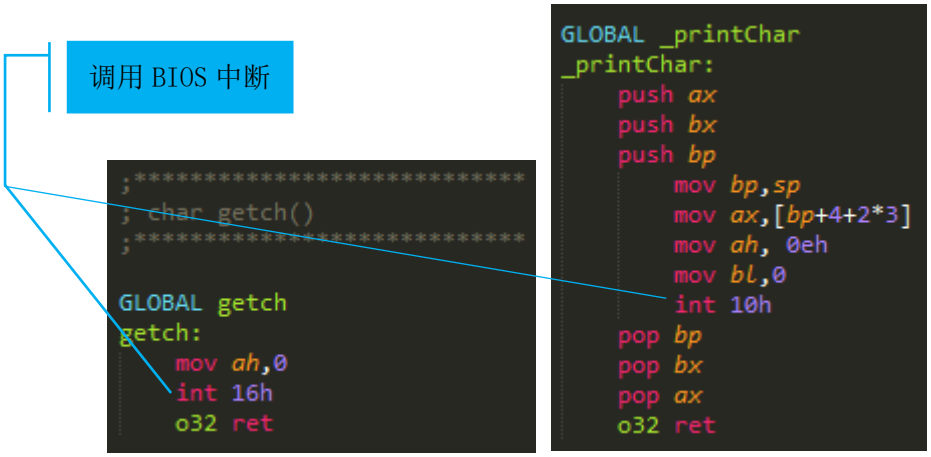


图 28 旧版非内核程序库输入输出

新版：

图 29 新版非内核程序库输入输出

可以看到，通过封装内核程序库、提供系统功能调用，C 程序库只需要简单地根据系统功能表设置功能号调用 21H 号中断。再者，由于在实验三中封装了 C 程序库以供 C 模块调用，接口不变，C 模块无所改动代码，这符合程序设计原则。

5. C 模块调用非内核程序库

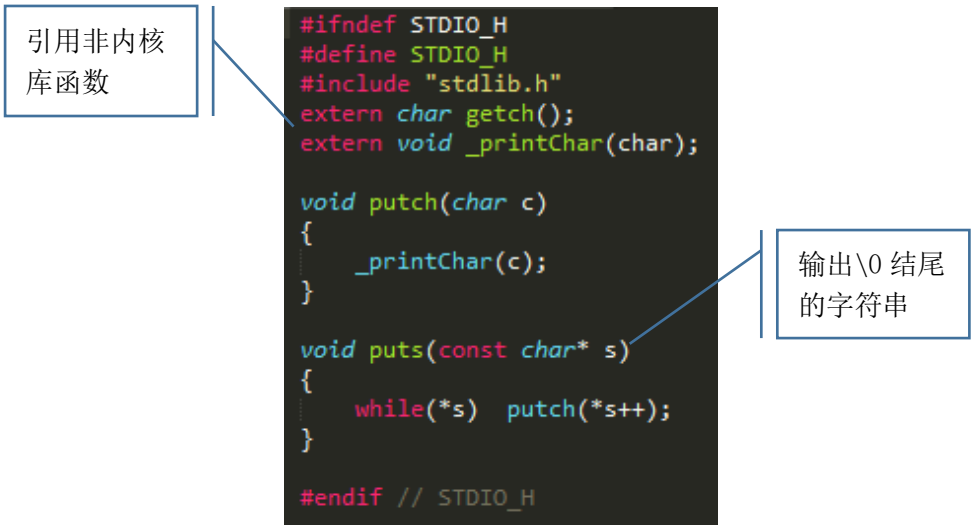


图 30 stdio.h

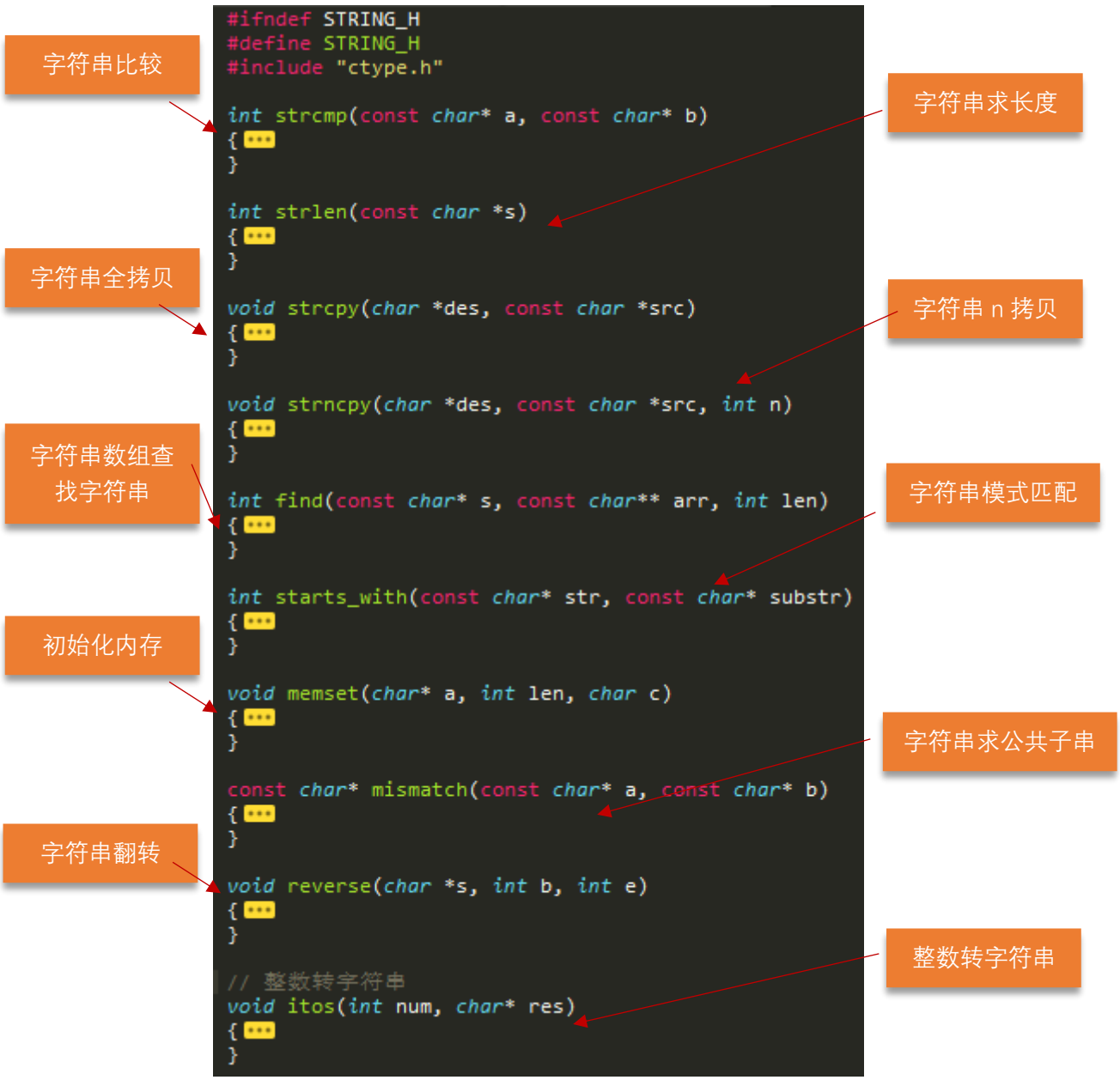


图 31 string.h

六、 【技术点与创新点】

(一) 音乐“键盘”

实现一个计算机钢琴“键盘”，运行用户程序时，按下 ASDFGHJK 键响起不同音符的声音。为了方便测试，设计了不返回的 play.com 程序。

电脑按键	A	S	D	F	G	H	J	K
音符	Do	Re	Mi	Fa	So	La	Si	Do(重音)

表格 2 电脑键盘与音符的对应

1. 在键盘中断中判断输入

通过“in al,60h”指令，用户按下的键的扫描码被读到 al 寄存器中。然后将

扫描码映射到对应音符的频率，接通扬声器控制其播放频率即可。

HEX	DEC	keys	HEX	DEC	keys	HEX	DEC	keys	HEX	DEC	keys
			10	16	Q	20	32	D	30	48	B	40	64	F6
01	1	ESC	11	17	W	21	33	F	31	49	N	41	65	F7
02	2	1	12	18	E	22	34	G	32	50	M	42	66	F8
03	3	2	13	19	R	23	35	H	33	51	,	43	67	F9
04	4	3	14	20	T	24	36	J	34	52	.	44	68	F10
05	5	4	15	21	Y	25	37	K	35	53	/	45	69	Num
06	6	5	16	22	U	26	38	L	36	54	R Shift	46	70	Scroll
07	7	6	17	23	I	27	39	;	37	55	PrtSc	47	71	Home
08	8	7	18	24	O	28	40	’	38	56	Alt	48	72	Up
09	9	8	19	25	P	29	41	`	39	57	Space	49	73	PgUp
0A	10	9	1A	26	[	2A	42	L Shift	3A	58	Caps	4A	74	-
0B	11	0	1B	27	]	2B	43	\	3B	59	F1	4B	75	Left
0C	12	-	1C	28		2C	44	Z	3C	60	F2	4C	76	Center
0D	13	=	1D	29	CTRL	2D	45	X	3D	61	F3	4D	77	Right
0E	14	bs	1E	30	A	2E	46	C	3E	62	F4	4E	78	+
0F	15	Tab	1F	31	S	2F	47	V	3F	63	F5	4F	79	End
50	80	Down												
51	81	PgDn												
52	82	Ins												
53	83	Del												

图 32 键盘扫描码与按键

每个频率占一个字（两个字节），因此下标要乘 2

```
music:
    in al,60h
    cmp al,1eh   ;'A'
    jb KeyboardEnd
    cmp al,25h   ;'K'
    jg KeyboardEnd
    sub al,1eh
    xor ah,ah
    shl al,1
    mov bx,ax
    push word[frequency+bx]
    call musick
    add sp,2
```

判断键盘输入是否合法

播放对应频率的铃声

图 33 判断键盘输入并映射到频率

2. 裸机响铃频率的控制

通过给 8253 定时器 2 装入不同的计数值，可使其输入不同频率的波形。与门打开后，经过放大器的放大作用，便可驱动扬声器发出不同的音调。要使该音调的声音持续一段时间，需插入一段延时，之后再将扬声器切断。

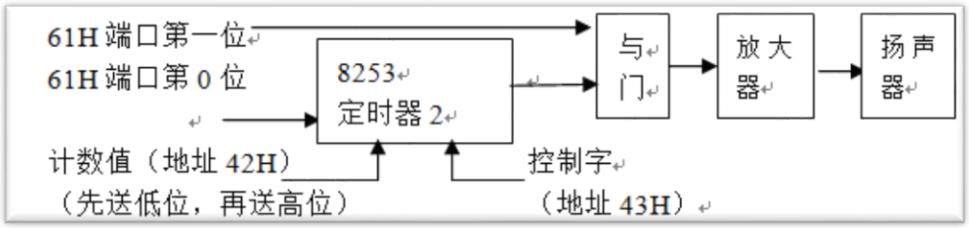


图 34 PC 机扬声器电路

3. 控制扬声器发声步骤

1) 计算频率值：以 120000H 为被除数，查频率表(已给出)中某项作为除数，所得商即为频率值

2) 设置控制字(8 位：10110110B)

假设此控制字八位由高到底 P7-P0,则：

P7P6 = 10: 选择 2 号计数器

P5P4 = 11： 读/写两个字节(先低位后高位)

P3P2P1 = 011: 选择计时器工作方式 3

P0 = 0: 设置计数器的工作码制为二进制

具体代码为：

```
MOV AL,10110110B;
OUT 43H,AL
```

3) 将频率值送入计数器

```
MOV AL,...
OUT 42H,AL;     设置计数器低 8 位
MOV AL,...
OUT 42H,AL;     设置计数器高 8 位
```

4) 打开与门

```
IN AL,61H;      取出 61H 端口数据
OR AL,03H;      设置 61H 端口最低两位为 11
```

5) 延时

6) 关闭与门

```
IN AL,61H;
AND AL,0FCH;        设置 61H 端口最低两位为 00
```

大体代码如下：

```
    in al, 61h
    or al, 3
    out 61h, al ;接通扬声器

    mov al, 0b6h    ;;将计数器2设为LSB和MSB,Model3,二进制格式
    out 43h, al
    mov dx, 12h
    mov ax, 348ch
    mov bp,sp
    add bp,2*5
    mov word bx,[bp+2]
    div word bx   ;音符的频率
    out 42h, al   ;给计时器2写入低位字节

    mov al, ah    ;给计时器2写入高位字节
    out 42h, al

    in al, 61h    ;获得61H端口当前的设置
    mov ah, al
    or al, 3      ;使PB0=1,PB1=以使扬声器可以发声
    out 61h, al   ;打开扬声器

    mov cx, 3314
push ax
;功能:产生一个N×15.08μs的时间延迟
;调用:(CX) = 15.08μs的倍数N
@waitf1:    ...
```

设置计时器 2 的属性来控制频率

图 35 频率控制、扬声器播放

七、 【实验总结】

这次实验，最大的收获无异于对中断和异步事件的理解。中断其实就是一种电信号，当它发生时，CPU 与对应中断端口之间产生脉冲，驱动处理器转移控制权。一个用户进程在任何时候、任何两条指令之间都可能被中断，即用户进程与中断之间是异步的，为了能正确响应中断并在中断程序执行完后能正确返回，便引出了保存现场的概念。通过

保存现场，把进程的寄存器、返回地址等存入内存，虚拟出一套 CPU，待中断返回时恢复现场，就实现了异步事件的捕捉、执行与返回。实际上，物理 CPU 只有一个，CPU 的控制权也只能在某一个进程之中，但“现场”的概念利用内存虚拟了一个 CPU，因此看起来好像是用户进程和中断处理程序是“同时”发生的。

当然，实验中也遇到了许多问题。一开始设计中断，是把它当成用户程序来做，即中断程序的代码与系统内核代码分处不同文件，通过编译中断程序的文件产生 com 文件，放在磁盘特定扇区，操作系统只要动态加载和运行中断程序就安装了中断。这个方法在设计时钟中断和键盘中断的时候工作得很好，而且添加新中断很方便，因为在之前的实验中加载运行用户程序已经封装地比较方便了。但是，到了软中断，问题就出现了。由于每个用户程序加载地址（CS）不一样，而为了能正确访问数据，需要将 DS 指向 CS，也就是说，每安装一个中断，DS、ES、SS 等的值都会改变，而稍有不慎，忘了恢复某些段寄存器，就会导致使用 int 指令调用中断时无法正确跳到中断处理逻辑中。这困扰了我一整天，最终也没找到到底哪里没有恢复段寄存器。于是，我果断放弃这种做法，直接把中断做成一个过程放在操作系统内核文件中，这样就不用每次安装中断都更改段寄存器的值，而是与操作系统共用段寄存器。

八、 【参考文献】

1. 实验四、五 PPT
2. 实验参考文献/PC 中断系统原理.doc
3. 汇编--计算机钢琴实现.CSDN Fivestar_wang